The journal for the international Ada community

Ada User Journal

Volume 41
Number 3
September 2020

Produced by Ada-Europe

Information on Subscriptions and Advertisements

Ada User Journal (ISSN 1381-6551) is published in one volume of four issues. The Journal is provided free of charge to members of Ada-Europe. Library subscription details can be obtained direct from the Ada-Europe General Secretary (contact details above). Claims for missing issues will be honoured free of charge, if made within three months of the publication date for the issues. Mail order, subscription information and enquiries to the Ada-Europe General Secretary.

For details of advertisement rates please contact the Ada-Europe General Secretary (contact details above).

ADA USER JOURNAL

Volume 41
Number 3
September 2020

Contents

Editorial Policy for Ada User Journal

Publication

Ada User Journal — The Journal for the international Ada Community — is published by Ada-Europe. It appears four times a year, on the last days of March, June, September and December. Copy date is the last day of the month of publication.

Aims

Ada User Journal aims to inform readers of developments in the Ada programming language and its use, general Ada-related software engine-ering issues and Ada-related activities. The language of the journal is English. Although the title of the Journal refers to the Ada language, related topics, such as reliable software technologies, are welcome. More information on the scope of the Journal is available on its website at *www.ada-europe.org/auj*.

The Journal publishes the following types of material:

Refereed original articles on technical matters concerning Ada and related topics.
Invited papers on Ada and the Ada standardization process.
Proceedings of workshops and panels on topics relevant to the Journal.
Reprints of articles published elsewhere that deserve a wider audience.
News and miscellany of interest to the Ada community.
Commentaries on matters relating to Ada and software engineering.
Announcements and reports of conferences and workshops.
Announcements regarding standards concerning Ada.
Reviews of publications in the field of software engineering.

Further details on our approach to these are given below. More complete information is available in the website at *www.ada-europe.org/auj*.

Original Papers

Manuscripts should be submitted in accordance with the submission guidelines (below).
All original technical contributions are submitted to refereeing by at least two people. Names of referees will be kept confidential, but their comments will be relayed to the authors at the discretion of the Editor.
The first named author will receive a complimentary copy of the issue of the Journal in which their paper appears.

By submitting a manuscript, authors grant Ada-Europe an unlimited license to publish (and, if appropriate, republish) it, if and when the article is accepted for publication. We do not require that authors assign copyright to the Journal.

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication else-where.

Proceedings and Special Issues

The *Ada User Journal* is open to consider the publication of proceedings of workshops or panels related to the Journal's aims and scope, as well as Special Issues on relevant topics. Interested proponents are invited to contact the Editor-in-Chief.

News and Product Announcements

Ada User Journal is one of the ways in which people find out what is going on in the Ada community. Our readers need not surf the web or news groups to find out what is going on in the Ada world and in the neighbouring and/or competing communities. We will reprint or report on items that may be of interest to them.

Reprinted Articles

While original material is our first priority, we are willing to reprint (with the permission of the copyright holder) material previously submitted elsewhere if it is appropriate to give it a wider audience. This includes papers published in North America that are not easily available in Europe.
We have a reciprocal approach in granting permission for other publications to reprint papers originally published in *Ada User Journal.*

Commentaries

We publish commentaries on Ada and software engineering topics. These may represent the views either of individuals or of organisations. Such articles can be of any length – inclusion is at the discretion of the Editor.

Opinions expressed within the *Ada User Journal* do not necessarily represent the views of the Editor, Ada-Europe or its directors.

Announcements and Reports

We are happy to publicise and report on events that may be of interest to our readers.

Reviews

Inclusion of any review in the Journal is at the discretion of the Editor. A reviewer will be selected by the Editor to review any book or other publication sent to us. We are also prepared to print reviews submitted from elsewhere at the discretion of the Editor.

Submission Guidelines

All material for publication should be sent electronically. Authors are invited to contact the Editor-in-Chief by electronic mail to determine the best format for submission. The language of the journal is English.

Our refereeing process aims to be rapid. Currently, accepted papers submitted electronically are typically published 3-6 months after submission. Items of topical interest will normally appear in the next edition. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.

Editorial

I would like to start this editorial by commenting on the delays that have been affecting the publication of the Ada User Journal during this year. This has been unfortunate, but justified by the exceptional pandemic situation that we have been living, which, among other effects, disrupted our well established printing process and somehow interfered with our plans concerning the contents to include in the journal, namely due to the cancellation of several events and hence the lack of material. We have been working to recover the delay and we believe that we will achieve this goal very soon. For certain, all the past issues (March and June) will be printed, and we expect that they will start arriving at your mailboxes early in 2021.

In this issue we are happy to include a special contribution prepared by Jeff Cousins, providing a very complete overview of Ada 202X, which is now almost completely finalised. This may serve as an excellent reference document for all those that use the language and follow its evolution and improvements to better suit developer needs and other requirements.

Then we also publish two articles that were accepted to the 20th International Real-Time Ada Workshop (IRTAW'2020). Although the event ended up being firstly postponed and then cancelled due to the COVID-19 situation, the authors of submitted papers were invited to prepare contributions to the AUJ, which we now bring to you. The first paper, authored by Luis Miguel Pinho (from the Polytechnic Institute of Porto, Portugal) and by Sara Royuela and Eduardo Quiñones (from the Barcelona Supercomputing Center, Spain), goes well with the special contribution referred above, as it addresses real-time issues in the mapping of parallel language features to the OpenMP tasking model, a possibility considered in Ada 202X (see Section 2 of that article). The second paper is by a team from the Polytechnic University of Madrid, including Jorge Garrido, David Pisonero, Juan Zamorano and Juan A. de la Puente, who analyse the support for vectorization existing in some programming languages and propose a possible extension to Ada for enhanced vectorization support.

As usual, this issue includes the Quarterly News Digest, prepared by Alejandro R. Mosteo, and the Calendar section, prepared by Dirk Craeynest. In addition to these two sections that have been an integral part of the AUJ for a long time, the issue includes the third puzzle prepared by John Barnes, which we may consider as a permanent section of the AUJ, as long as John Barnes will be willing to entertain and surprise us with his challenges. This time, the puzzle is about nested squares, and is not an easy one! The solution to the Greek Cross problem brought to you in the previous issue is also provided.

We close this editorial with the sad note that Ian Christopher Wand left us last July. Although I did not personally know Ian Wand, I learned that he was a very reputed computer scientist, recognized by his peers, with important contributions to the Ada community. This issue includes an In memoriam prepared by Ian Pyle and other colleagues.

Antonio Casimiro
Lisboa
September 2020
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Alejandro R. Mosteo

Centro Universitario de la Defensa de Zaragoza, 50090, Zaragoza, Spain; Instituto de Investigación en Ingeniería de Aragón, Mariano Esquillor s/n, 50018, Zaragoza, Spain; email: amosteo@unizar.es

Contents

[Messages without subject/newsgroups are replies from the same thread. Messages may have been edited for minor proofreading fixes. Quotations are trimmed where deemed too broad. Sender's signatures are omitted as a general rule. --arm]

Preface by the News Editor

Dear Reader,

This number brings again important news for Open Source enthusiasts from the GNAT front. A survey was conducted by AdaCore to gather feedback on the idea of discontinuing the Community Editions in favor of better supporting the FSF-maintained version packaged by various Linux distributions. You can read reactions about this idea in thread [1].

The Jupyter notebooks for Ada by Maxim Reznik (first reported here in AUJ 41.2) are quickly taking shape: a series of interactive tutorials demonstrating new features of Ada 202x is already available on-line, with 10 entries at the time of this writing. Find more about these in [2], and of course visit them with your browser to witness their potential first-hand.

If you would like to go down memory lane, two threads about operating systems (supporting or implemented in Ada) contain juicy bits in the Ada and Operating Systems section. Or, if you prefer to look forward to hypothetical future Ada features, a large discussion emerged from the embers of an old thread proposing solutions to the automatic storage of indefinite types [3].

I ask for your indulgence for closing this preface with a project I started and actively develop (in collaboration, chiefly, with Fabien Chouteau from AdaCore): Alire (after Ada Library Repository), a package manager for Ada and SPARK has entered public beta, and debuts in this issue [4]. As of this writing, Alire indexes 130 libraries and executable projects that you can immediately retrieve and build with GNAT without a care in the world about having to go hunting for dependencies. (A technical paper about an early version of Alire was published in AUJ 39.3.)

Sincerely,
Alejandro R. Mosteo.

[1] "Survey on the Future of GNAT Community Edition", in Ada Practice.

[2] "Ada 2020 Jupyter Notebooks", in Ada and Education.

[3] "Proposal: Auto-allocation of Indefinite Objects", in Ada Practice.

[4] "Repositories of Open Source Software", in Ada-related Resources.

Ada and Education

Ada 2020 Jupyter Notebooks

From: Maxim Reznik <reznikmm@gmail.com>
Subject: Ada 2020 Jupyter notebooks
Date: Wed, 2 Sep 2020 06:28:07 -0700
Newsgroups: comp.lang.ada

I'm going to write a series of Jupyter notebooks about Ada 2020 support in GNAT Community Edition 2020.

First two are there:

- Ada 2020: 'Image attribute for any type
- Ada 2020: Redefining the 'Image attribute

https://github.com/reznikmm/ada-howto/tree/ce-2020

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Wed, 2 Sep 2020 23:49:46 -0700

Nice idea, this is a nice way to teach Ada indeed.

New(?) Intros to Ada and Spark on adacore.com

From: Paul Rubin <no.email@nospam.invalid>
Subject: new(?) intros to Ada and Spark on adacore.com
Date: Wed, 02 Sep 2020 23:50:01 -0700
Newsgroups: comp.lang.ada

I don't remember seeing these here before. They look promising: https://learn.adacore.com/courses/courses.html:

- Introduction to Ada
- Introduction to SPARK
- Ada for the C++ or Java Developer
- SPARK Ada for the MISRA C Developer
- Introduction to GNAT Toolchain

https://learn.adacore.com/courses/intro-to-ada/index.html

https://learn.adacore.com/courses/intro-to-spark/index.html

https://learn.adacore.com/courses/Ada_For_The_CPP_Java_Developer/index.html

https://learn.adacore.com/courses/SPARK_for_the_MISRA_C_Developer/index.html

https://learn.adacore.com/courses/GNAT_Toolchain_Intro/index.html

Solutions to J. McCormick Book

From: Werner Aeschbacher <aeschbaw@ieee.org>
Subject: Training Ada
Date: Wed, 23 Sep 2020 11:43:08 -0700
Newsgroups: comp.lang.ada

Does anybody have the solutions to the exercises of the book "Building Parallel, Embedded, and Real-Time Applications with Ada" from John W. McCormick et al ?

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 23 Sep 2020 12:58:45 -0700

Is there a claim that a solution set was published someplace? Your best bet might be to contact the authors.

Sometimes with textbooks (say in mathematics), there is a solutions book available only to instructors, so they can assign homework problems from the textbook and check students' answers against the solutions book.

I have the textbook you mention. It looks good but I haven't gotten around to reading much of it. If there's a particular exercise you're interested in, I might like to give it a try.

From: "Randy Brukardt" <randy@rrsoftware.com>
Date: Wed, 23 Sep 2020 16:37:52 -0500

> Your best bet might be to contact the authors.

Agreed. John McCormick is still involved in Ada (he was on an ARA meeting this morning), so I'd expect he'd be able to give you some information.

Solutions to J. English Book

From: Jack Davy <jules1.davy@gmail.com>
Subject: Learning Ada
Date: Tue, 15 Sep 2020 03:36:34 -0700
Newsgroups: comp.lang.ada

I've just started learning Ada and am using the book "Ada95: The Craft of Object Oriented Programming", by John English. I know there are plenty of other resources such as the one on AdaCore, which covers Ada 2012, but I like the style and flow of this book. Anyway, I was wondering whether anyone in the group has the answers to the end of chapter exercises? The author has now retired and the link to them is dead.

Thanks in Advance!

From: Anders Wirzenius <anders.wirzenius@netikka.fi>
Date: Tue, 15 Sep 2020 17:31:57 +0300

Maybe this helps:

http://archive.adaic.com/docs/craft/craft.html

From: Jack Davy <algojack@tutanota.com>
Date: Tue, 15 Sep 2020 08:07:07 -0700

Thanks Anders, but I already found that link. The download has the code for the book, but no answers. I guess it's not important, I just thought it would be nice to see some sample solutions.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 15 Sep 2020 17:54:35 +0200

I don't have an answer to your exact question but there is no shortage of "sample solutions" in Ada on https://rosettacode.org/wiki/Category:Ada

HTH

PS. I still consider John English's book to be the best introduction to Ada.

From: Simon Wright <simon@pushface.org>
Date: Tue, 15 Sep 2020 18:01:56 +0100

Try here: https://www.dropbox.com/s/8k4xxpj5a67s752/adacraft.tar.gz?dl=0

Nothing like being a pack rat! My hard disk copy is dated 2012-8-25, but I don't know when I retrieved it, must have been several computers ago. Internal dates up to 2001-07-27. Readme says examples tested with GNAT 3.13p!

From: Simon Wright <simon@pushface.org>
Date: Tue, 15 Sep 2020 18:07:13 +0100

Actually, they are at adaic.com: http://archive.adaic.com/docs/craft/craft.html, see the third bullet point.

From: Jack Davy <algojack@tutanota.com>
Date: Tue, 15 Sep 2020 12:03:46 -0700

@ Ludovic, thanks for the link to rosettacode; very good source of examples. And good to hear that you rate the book highly. There don't seem to be many books on Ada, but there is a very recent one for beginners which I will probably get to fill in the gaps not covered by "The Craft". https://www.apress.com/gp/book/9781484254271

@ Simon, thanks, but I already have that file. It contains all the code in the book but not the answers to the end of chapter questions.

By the way, I see the author also wrote a GUI library for Ada called JEWL, the files for which I have also downloaded. Pity it's for Windows only. I'm a Linux user although I do have Win XP on VirtualBox, but I don't believe the current GNAT compiler will run on it.

From: Gautier write-only <gautier_niouzes@hotmail.com>
Date: Tue, 15 Sep 2020 12:28:20 -0700

Other sample sources:

Ada resources:

- https://sourceforge.net/directory/language:ada/
- https://www.adaic.org/ada-resources/

Small samples are embedded in the LEA editor (you can run it from Wine):

https://sourceforge.net/projects/l-e-a/

From the menu: Action / Code sample. Choose your sample. Hit F9 for running.

Some samples stem from Rosetta Code BTW :-)

From: Jerry Petrey <gpetrey@cox.net>
Date: Tue, 15 Sep 2020 16:00:08 -0700

> By the way, I see the author also wrote a GUI library for Ada called JEWL [...]

Yes, his JEWL package is great. I used it many times to create Windows GUI apps and still use it some. I talked to John a number of times - he was very helpful. His book is one of the best!

From: Paul Rubin <no.email@nospam.invalid>
Date: Tue, 15 Sep 2020 18:23:18 -0700

> [...] there is a very recent one for beginners which I will probably get to fill in the gaps not covered by "The Craft".https://www.apress.com/gp/book/9781484254271

I haven't examined that book directly but based on the preview and blurb, it does seem to be beginner oriented, thus likely to have gaps of its own. If you're trying to fill gaps, you probably want something more complete and advanced.

I semi-recently got another book that looks very good, though it's still sitting around without my having read much of it: Analysable Real-Time Systems: Programmed in Ada, by Andy Wellings and Alan Burns. It is basically an updated reprint of an older book by the same authors, self-published in paperback, so it is a good value.

From: Jack Davy <algojack@tutanota.com>
Date: Wed, 16 Sep 2020 00:13:22 -0700

@ Gautier, thanks for the links. When I get Windows 7 on VirtualBox I'll give the LEA editor a try, I'm not so keen on using Wine, it's a bit hit & miss. Also since I learned Vim a few years ago no other editors really do it for me, unless they have Vim bindings ;).

@ Paul, I was thinking that the beginner's Apress book would fill in the gaps regarding Ada 2012 specifically, which as I understand it has changed from previous versions mainly in regard to OOP; I'm assuming I won't need to unlearn anything if I learn the basics from an Ada 95 book. The real-time stuff would be over my head at this point I think, and not really something I had in mind when considering Ada, although I do have a background in electronics, and see that there is Ada compiler for AVR on AdaCore.

The more I look at this language the more I wonder why it isn't more popular. Maybe people just don't like the pascalish syntax, but that never put me off because I learned Turbo Pascal at Uni (25 years ago) and more recently Free Pascal/Lazarus. Never was much of a fan of the curly bracket languages.

From: Jack Davy <algojack@tutanota.com>
Date: Wed, 16 Sep 2020 00:32:32 -0700

I found an impressive list of 'Things to like about Ada' posted by a C/C++ career programmer on the AVR freaks forum (in reply #13) :

https://www.avrfreaks.net/forum/i-didnt-know-you-could-get-ada-avr

My main reason for wanting to learn Ada is the last on his list: "Promotes a professional, anti-hacker mentality. By being unforgiving the language promotes the valuable discipline of specifying and writing code more exactly, without the temptations of slipping into bit-twiddling or other programming habits that subvert (and often break) the data or code models. When proper programming discipline is not enforced by the language then it must be voluntary, and in those cases discipline can and inevitably will slip, but when the language enforces much of that discipline then there are no easy ways to avoid it, and the resulting code is higher in quality and faster to develop."

Maybe that's why Ada isn't more popular - being disciplined isn't easy, and hacking is more fun. But I've learned the hard way that it's actually much more satisfying when your programs are bug-free and work properly the first time you run them. Any language which enforces more thinking and less trial-and-error coding is a winner in my book.

From: Gautier write-only <gautier_niouzes@hotmail.com>
Date: Wed, 16 Sep 2020 02:13:54 -0700

> @ Gautier, thanks for the links. When I get Windows 7 on VirtualBox I'll give the LEA editor a try, I'm not so keen on using Wine, it's a bit hit & miss.

No worries, you can access the same samples (and the same compiler) without LEA, built on your preferred operating system.

>- https://hacadacompiler.sourceforge.io/ (source code here: https://sourceforge.net/p/hacadacompiler/code/HEAD/tree/, mirrored here: https://github.com/zertovitch/hac)

Mutatis mutandis, you get there the "tpc.exe" equivalent, whereas LEA is the "turbo.exe" :-)

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 16 Sep 2020 12:55:58 +0200

> The more I look at this language the more I wonder why it isn't more popular. [...]

I wasn't there when it happened but I read that early Ada 83 compilers were buggy, slow and outrageously expensive because marketed only at one captive customer, the US DoD. (In their defence, Ada is a particularly difficult language to implement well, orders of magnitude more so than Pascal or C). The vendors never really tried to sell Ada development tools outside the military, despite hype that Ada was the language of the future. At around the same time, C++ used the opposite strategy of selling cheap compilers, with the additional advantage of backward compatibility with C, so they won market share. Turbo Pascal was a contender back then but only on DOS and Windows, so it ultimately lost to C++, possibly in no small part because of Borland's refusal to abide by any portable standard. And then Sun marketed Java aggressively with a zero-cost compiler and promises of ultimate portability, and stole the show.

The Ada landscape changed dramatically when the first Free Software Ada 95 compiler, GNAT, arrived, but the damage to the reputation of Ada was very hard to overcome. An entire generation of military and corporate programmers, frustrated by the early compilers, became managers and dismissed Ada out of hand for decades. They and their prejudices have started to retire in the past few years and I think this is one factor in the current renaissance of Ada.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Wed, 16 Sep 2020 13:09:54 +0200

> I wasn't there when it happened but [...]

I mostly agree with your analysis, except the last part. The problem is that the culture of programming and overall education became so low that it is no more a race against C++. C++ itself is in defense and losing against languages and practices so overwhelmingly bad that even C looks as a shining beacon. Winter is coming.

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Date: Wed, 16 Sep 2020 17:01:41 +0200

> The more I look at this language the more I wonder why it isn't more popular.

Ada is a language for engineering software. Since 98% of developers are unable to do that, Ada will never be popular as long as such people are allowed to develop software.

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 16 Sep 2020 14:29:56 -0700

> @ Paul, I was thinking that the beginner's Apress book would fill in the gaps regarding Ada 2012 specifically, which as I understand it has changed from previous versions mainly in regard to OOP

I think Ada 95 OOP is not really used very much, and the changes in Ada 2012 are things like contracts, and built-in SPARK syntax. You could also look at the online book "Ada Distilled" which is about Ada 95. I found it an ok way to get started, though I never really progressed beyond that.

> I do have a background in electronics, and see that there is an Ada compiler for AVR on AdaCore.

I don't know the current state of that, but some years ago it was rather hard to use or parts were missing or whatever. These days, the AVR is in decline since it is so limited. Everyone uses ARM or maybe soon RISC-V processors even for tiny embedded stuff.

> The more I look at this language the more I wonder why it isn't more popular. Maybe people just don't like the pascalish syntax

Tooling, libraries, language verbosity, etc. As pure language, though, it is still mysterious to me what Rust offers that Ada doesn't.

Today, for most programming, "systems languages" including Ada, C, C++, and Rust are all imho somewhat niche. Unless you are dealing with specialized problems (such as embedded or OS's), computers have almost unbounded resources. So it's easier to get your work done using languages with automatic memory management, unbounded arithmetic, etc.

The main cost is consuming more machine resources and losing some timing determinism, but most of the time you can live with both of those. Ada is best for more demanding applications which usually involve realtime or high reliability constraints.

From: Mart van de Wege <mvdwege@gmail.com>
Date: Fri, 18 Sep 2020 08:53:20 +0200

> Ada is a language for engineering software. [...]

I use it for hobby stuff, for quick solutions (like generating RPG characters). Does not feel like engineering to me.

But what I do like is the elegance of the language, and the ability to describe my problem domain using distinct types.

The 'verbosity' does not bother me. I'm a fluent touch typist, Using the shift key to type braces slows me more than typing out statements to delineate blocks.

The only real nit I have with Ada is that it does not have closures.

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Date: Fri, 18 Sep 2020 12:00:47 +0200

> I use it for hobby stuff, for quick solutions (like generating RPG characters). Does not feel like engineering to me.

I do similar things, too, but I always have a design in mind, and usually start with pkg, task, & PO specs and subprogram declarations, so I suspect that after doing this for so long I can engineer simple problems in my head. Presumably others with similar experience or who are better than I do the same.

Ada-related Resources

[Delta counts are from Apr 6th to Jul 20th. --arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: Mon, 02 Nov 2020 18:41:21 +0100
To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn: 3_025 (+75) members	[1]
- Reddit: 4_720 (+634) members	[2]
- Stack Overflow: 1_924 (+60) questions	[3]
- Freenode: 90 (+2) users	[4]
- Gitter: 64 (+8) people	[5]
- Telegram: 90 (+11) users	[6]
- Twitter: 67 (+14) tweeters	[7]
92 (+27) unique tweets	[7]

[1] https://www.linkedin.com/groups/114211/

[2] http://www.reddit.com/r/ada/

[3] http://stackoverflow.com/questions/tagged/ada

[4] https://netsplit.de/channels/details.php?room=%23ada&net=freenode

[5] https://gitter.im/ada-lang

[6] https://t.me/ada_lang

[7] http://bit.ly/adalang-twitter

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: Mon, 02 Nov 2020 18:41:21 +0100
To: Ada User Journal readership

[This issue sees a newcomer, the Alire package manager project, debuting with 130 Ada projects ready to use. --arm]

Rosetta Code: 747 (=) examples	[1]
37 (=) developers	[2]
GitHub: 729 (+77) developers	[3]
Sourceforge: 276 (+1) projects	[4]
Open Hub: 212 (=) projects	[5]
Alire:130 (new!) crates	[6]
Bitbucket: 88 (-2) repositories	[7]
Codelabs: 52 (+1) repositories	[8]
AdaForge: 8 (=) repositories	[9]

[1] http://rosettacode.org/wiki/Category:Ada

[2] http://rosettacode.org/wiki/Category:Ada_User

[3] https://github.com/search?q=language%3AAda&type=Users

[4] https://sourceforge.net/directory/language:ada/

[5] https://www.openhub.net/tags?names=ada

[6] https://alire.ada.dev/crates.html

[7] https://bitbucket.org/repo/all?name=ada&language=ada

[8] https://git.codelabs.ch/?a=project_index

[9] http://forge.ada-ru.org/adaforge

Language Popularity Rankings

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada in language popularity rankings
Date: Mon, 20 Jul 2020 09:38:21 +0100
To: Ada User Journal readership

[From this number on, positive ranking changes mean to go up in the ranking. This issue sees the addition of the PYPL ranking, which is computed by analyzing how often language tutorials are searched on Google. The IEEE ranking has seen no updates through 2020, and will be likely dropped soon if this situation persists. --arm]

- TIOBE Index: 39 (+4) 0.35% (+0.07%)	[1]
- PYPL Index: 19 (new!) 0.62% (+0.3%)	[2]
- IEEE Spectrum (general): 43 (=) Score: 24.8	[3]
- IEEE Spectrum (embedded): 13 (=) Score: 24.8	[3]

[1] https://www.tiobe.com/tiobe-index/

[2] http://pypl.github.io/PYPL.html

[3] https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2019

Ada Reference Manual 2020.1

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Ada Reference Manual info format 2020.1 released.
Date: Fri, 17 Jul 2020 10:23:15 -0700
Newsgroups: comp.lang.ada

ada-ref-man 2020.1 is now available in GNU ELPA.

This includes Ada 202x draft 25, as well as Ada 2012. GNAT Community 2020 has some support for some of the new language features in Ada 202x.

There is also now a searchable info index, containing the entries in the ARM Index.

From: "Randy Brukardt" <randy@rrsoftware.com>
Date: Sat, 18 Jul 2020 22:40:16 -0500

Sounds good, but keep in mind this is a moving target. Draft 26 should be available next week. :-)

From: "Randy Brukardt" <randy@rrsoftware.com>
Date: Fri, 31 Jul 2020 17:55:47 -0500

> Sounds good, but keep in mind this is a moving target. Draft 26 should be available next week. :-)

My primary computer died (now fixed, knock on wood), and we've since had an ARG meeting, so this new draft will be delayed a couple of weeks. Shouldn't be too far in the future, though.

Ada-related Tools

SweetAda 0.1C-0.1F

From: gabriele.galeotti.xyz@gmail.com
Subject: SweetAda 0.1c released
Date: Tue, 7 Jul 2020 14:30:36 -0700
Newsgroups: comp.lang.ada

I've just released SweetAda 0.1c.

Windows toolchains now have libstdc++ included.

The RISCV32 and RISCV64 toolchains are now deprecated, because they end up the same. So there is now a generic RISCV toolchain. It behaves like the other two, you have just to specify the correct CPU. GCC switches that activate the 64-bit mode are "-march=rv64imafdc" and "-mabi=lp64d". Obviously there is the correspondent RTS target.

The RISCV support is a little bit usable, if you pick the QEMU-RISC-V-32 platform, it runs Ada code and does some primitive I/O in the IOEMU window, stimulating a LED and an 8-bit port.

I've tested Insight and it works very well, breakpoints and other things seem ok.

Other minor adjustments here and there.

I saw in the log that many users still try to download from a non-existent directory, i.e., sweetada.org/software/.... Please update your links, the correct directory is sweetada.org/packages/....

Thanks for your patience, I am also working on documentation.

From: gabriele.galeotti.xyz@gmail.com
Subject: SweetAda 0.1e released
Date: Wed, 22 Jul 2020 11:03:04 -0700
Newsgroups: comp.lang.ada

Hi all. I've just released SweetAda 01.e.

Go to http://www.sweetada.org and download the archive.

RTS and LibGCC packages are still valid @ 0.1c.

- general cleanup and cosmetics
- general infrastructure improvements
- QEMU-RISC-V-32 target can do serial output in a terminal
- IntegratorCP target uses LCD VGA
- Malta MIPS target uses a VGA PCI board
- handling of directories in the cpus hierarchy, which allows selective unit overriding
- Insight can be called as a toolchain component
- IOEMU configuration files are now fully consistent

Next days I will concentrate on generic low-level CPU support, documentation, and restructuring of some redundant units. Let me know, feedback is highly appreciated.

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: SweetAda 0.1f released
Date: Wed, 19 Aug 2020 15:40:58 -0700
Newsgroups: comp.lang.ada

Hi all, I've just released the 0.1f version of SweetAda.

- general cleanup and cosmetics
- general infrastructure improvements
- the VGA text driver is now unified across platforms; it is actually used by PC-x86, PC-x86-64 and MIPS Malta
- the ugly handling of network packets (Amiga/FS-UAE and PC-x86) is re-routed to a PBUF FIFO handler (the management is still far from ideal, but is not tied to the ISR like before)
- various I/O have now correct aspect specifiers; in particular some hardware registers with specific sizes are now correctly handled without premature optimizations
- AVR is now part of SweetAda and so 2 platforms exist: ArduinoUno and a QEMU emulator (both ATmega328P);

  the AVR support is primitive and incomplete, but, with an ArduinoUno board, is sufficient to start up the Ada infrastructure and is able to pulse the onboard LED;

  note that programming is performed by means of the AVRDUDE tool, so you should use a version suitable for your environment;

  otherwise you could use the IHEX .hex output file with your preferred tool;

  the QEMU-AVR platform can be used with GDB or Insight to trace the execution of code;
- runsweetada and IOEMU library now correctly show in argv dumps the launched executable instead of a "NULL" tag
- the parser inside the IOEMU library now expose in the .cfg file a variable (LASTPID) that carries the PID of the last launched executable (see QEMU-AVR/qemu.cfg)

There is also a new release of all QEMU emulators, at version 5.1.0.

Please note that the Linux version is linked with the SDL2 library instead of the previous GTK+3.

You can find everything at http://www.sweetada.org

GWindows 31-Jul-2020

From: gautier_niouzes@hotmail.com
Subject: Ann: GWindows release, 31-Jul-2020
Date: Fri, 31 Jul 2020 11:01:11 -0700
Newsgroups: comp.lang.ada

GWindows is a full Microsoft Windows Rapid Application Development framework for programming GUIs (Graphical User Interfaces) with Ada.

GWindows works with the GNAT development system (could be made pure Ada with some effort).

Changes to the framework are detailed in gwindows/changes.txt or in the News forum on the project site.

In a nutshell (since last announcement here):

- a few features from the extensions GWindows.Common_Controls. Ex_List_View and GWindows.Common_Controls. Ex_TV_Generic have been moved to parent package and respective parent types for broader use
- fix: a few records for binding with the Windows API were erroneously 32-bit only

GWindows Project site: https://sf.net/projects/gnavi/

GWindows GitHub clone: https://github.com/zertovitch/gwindows

TASH Sources

From: mockturtle <framefritti@gmail.com>
Subject: TASH sources?
Date: Mon, 3 Aug 2020 02:16:15 -0700
Newsgroups: comp.lang.ada

I wanted to try to use the Ada Tcl/Tk binding TASH [1], but the download page has links to www.adatcl.com that sends me to some chinese-written site. I guess www.adatcl.com expired?

Does someone know where I can find the sources of TASH?

Please note that, for reasons too long to be explained here, I am not interested in alternatives to TASH, unless they are Tcl/Tk bindings.

Thank you in advance.

[1] http://tcladashell.sourceforge.net/index.htm

From: mockturtle <framefritti@gmail.com>
Date: Mon, 3 Aug 2020 02:28:20 -0700

I am replying to my own post... Deep down in the Google results I found a github version of TASH

https://github.com/simonjwright/tcladashell

Despite the different name it seems like the original sourceforge TASH (or a fork?) revived on github

From: Simon Wright <simon@pushface.org>
Date: Mon, 03 Aug 2020 14:38:58 +0100

> Do someone know where I can find the sources of TASH?

I altered the project page on SF to point to the new Github site, but forgot about the project web pages. Sorry.

I've been moving my projects to Github; it's a far more pleasant and performant environment, I find.

> [1] http://tcladashell.sourceforge.net/index.htm

This page now points you to https://github.com/simonjwright/tcladashell

From: Simon Wright <simon@pushface.org>
Date: Mon, 03 Aug 2020 14:40:27 +0100

> Despite the different name it seems like the original sourceforge TASH (or a fork?) revived on github

I thought it was the same name?

Anyway, the project has moved to Github, under the same management :-)

From: gautier_niouzes@hotmail.com
Date: Mon, 3 Aug 2020 06:53:18 -0700

There is also on GitHub: https://github.com/thindil/tashy ("TASHY is short from Tcl Ada SHell Younger").

SI Units Checked and Unchecked

From: AdaMagica <christ-usch.grein@t-online.de>
Subject: SI Units Checked and Unchecked - Completela overhauled version
Date: Thu, 13 Aug 2020 05:24:06 -0700
Newsgroups: comp.lang.ada

Simplified design now available:

http://archive.adaic.com/tools/CKWG/Dimension/SI.html

The choice of using dimension checking or not is now made via a generic signature package. The user interface is unchanged.

Resource to Source

From: <s@srin.me>
Subject: Ann: Resource to source
Date: Thu, 27 Aug 2020 12:09:17 -0700
Newsgroups: comp.lang.ada

Tool "resource" is available - (MIT License) at: https://gitlab.com/cpp8/bindata

This tool can take resource files, graphics, audio etc. and convert them into Ada (or C) source code and it can be compiled and included in the binary.

Developed for pedagogic reasons (https://github.com/RajaSrinivasan/assignments/blob/master/resource.pdf) but hoping it will be useful to the community.

Simple Components 4.51

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components for Ada 4.51 IEEE 754-2008 Decimal
Date: Mon, 31 Aug 2020 15:28:14 +0200
Newsgroups: comp.lang.ada

The current version provides implementations of smart pointers, directed graphs, sets, maps, B-trees, stacks, tables, string editing, unbounded arrays, expression analyzers, lock-free data structures, synchronization primitives (events, race condition free pulse events, arrays of events, reentrant mutexes, deadlock-free arrays of mutexes), pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support, streams, multiple connections server/client designing tools and protocols implementations. The library is kept conform to the Ada 95, Ada 2005, Ada 2012 language standards.

http://www.dmitry-kazakov.de/ada/components.htm

Changes (1 September 2020) to the version 4.50:

- The HTTP client behavior changed not to close connection when keep alive flag is set unless the server explicitly requests closing it;
- Non-standard request headers added to the HTTP implementation: X-Requested-By, X-Requested-With, X-XSRF-TOKEN, X-CSRF-TOKEN;
- The package IEEE_754.Decimal32 was added. The package implements IEEE 754-2008 decimal32 format;
- The package IEEE_754.Decimal64 was added. The package implements IEEE 754-2008 decimal64 format;
- The package IEEE_754.Decimal128 was added. The package implements IEEE 754-2008 decimal128 format;
- An implementation of 128-bit integers was added to the package IEEE_754;
- The package IEEE_754.Edit was added;
- The package provides strings formatting facilities for 128-bit integers;
- Fallback time zone names changes in the package GNAT.Sockets. Connection_State_Machine. ELV_MAX_Cube_Client.Time_Zones.

Image_Random

From: PragmAda Software Engineering <pragmada@pragmada. x10hosting.com>
Subject: [Ann] Image_Random
Date: Thu, 3 Sep 2020 17:17:03 +0200
Newsgroups: comp.lang.ada

Image_Random: True random numbers from a digital camera (under Linux with the GNAT compiler) is now available in case anyone finds it useful.

https://github.com/jrcarter/Image_Random

MP Music Player

From: PragmAda Software Engineering <pragmada@pragmada. x10hosting.com>
Subject: [Ann] MP
Date: Tue, 15 Sep 2020 22:11:30 +0200
Newsgroups: comp.lang.ada

MP, a Music Player based on the Gnoga audio widget, is available at https://github.com/jrcarter/MP

Ada-related Products

PTC ObjectAda V10.2 for Windows

[This PTC announcement and the following companion were already published in the previous AUJ number, although by date they properly belong in this number, so here they are again. --arm]

From: Shawn Fanning <sfanning@ptc.com>
Subject: Product Release Announcement – PTC ObjectAda V10.2 for Windows
Date: Mon, 27 Jul 2020 16:39:05 -0700
Newsgroups: comp.lang.ada

On July 22, 2020, PTC announced the availability of version 10.2 of our ObjectAda for Windows and ObjectAda64 for Windows products. This new product release provides full support for Ada 2012 language features and represents the completion of the phased implementation strategy PTC adopted for Ada 2012 language feature support within the ObjectAda technology. With ObjectAda for Windows version 10.2, the ObjectAda compiler conforms to the Ada Conformity Assessment Test Suite (ACATS) version 4.1Q and adds several new features not present in the previous release (ObjectAda version 10.1 released in May 2019) including support for storage subpools and the Default_Storage_Pool pragma, execution time enforcement of type invariants, and complete support for new Ada expression forms.

The new installation approach introduced with ObjectAda for Windows v10.x allows ObjectAda to be used with the latest releases of Microsoft's Visual Studio tools and the Windows 10 SDK. ObjectAda version 10.2 includes version 4.0.0 of the ObjectAda Ada Development Toolkit (ADT) Eclipse interface which supports Eclipse 2020-03 (4.15) or later. All of these upgrades combined make ObjectAda for Windows version 10.2 a solid, modern, and effective toolset for development of mission-critical application code in the Ada language. ObjectAda version 10.2 supports Ada 95, Ada 2005, and Ada 2012 compiler operation modes to provide compatibility with previous versions.

Additional information about ObjectAda version 10.2 is available within the Product Release Announcement which can be downloaded from https://www.ptc.com/products/developer-tools/objectada.

Customers with active subscription licenses for ObjectAda for Windows v10.x or ObjectAda64 for Windows v10.x are entitled to a no-charge upgrade to v10.2.

If you are not currently using ObjectAda and wish to learn more or if you are using an earlier release of ObjectAda and wish to upgrade, register your request at https://www.ptc.com/en/products/developer-tools/objectada/contact-sales.

PTC ApexAda V5.2 Embedded for Linux/ARMv8 64-bit

From: Shawn Fanning <sfanning@ptc.com>
Subject: Product Release Announcement – PTC ApexAda v5.2 Embedded for Linux/Armv8 64-bit
Date: Mon, 27 Jul 2020 16:42:37 -0700
Newsgroups: comp.lang.ada

On May 19, 2020 PTC announced the release of the PTC ApexAda v5.2 Embedded for Linux/Armv8 64-bit product. This product is the initial product offering based on a new 64-bit code generator for ApexAda for the Armv8 64-bit (aarch64) architecture and is our latest release supporting 64-bit embedded application development.

The host operating system for this product is Intel x64 Red Hat Enterprise Linux v7.x/v8.x (or CentOS equivalent) distribution. Using the Linaro GNU cross-development toolchain for 64-bit Armv8 Cortex-A processors on the Linux/Intel64 host, PTC ApexAda supports the generation of Ada 95 / Ada 2005 application images that execute on ARMv8-A 64-bit (aarch64) processors (for example Arm Cortex A53, A57, A72) running 64-bit embedded Linux distributions. Examples of embedded Linux distributions which can be supported are openSUSE Leap v15.1, SUSE Linux Enterprise Server for Arm v15.1, Ubuntu Server 20.04, Wind River Linux and other Yocto-derived Linux distributions with a 64-bit kernel. Reference hardware used for the development and test of ApexAda was the Raspberry Pi 3 Model B/B+. (Raspberry Pi 4 Model B with its larger 4GB RAM configuration and other boards such as the VPX-1703 from Curtiss-Wright Defense Solutions can also be supported by ApexAda.)

Included with the 64-bit embedded compiler is the PTC® ApexAda v5.2 64-bit compiler for Linux native application development. Also included is the integrated ApexAda 64-bit C/C++ compiler which facilitates seamless development of mixed-language applications written in Ada, C, and C++. ApexAda V5.2 Embedded compilers provide a complete cross-development toolchain hosted from Linux distributions including RedHat Enterprise Edition, CentOS, and SUSE. A complete description of PTC ApexAda v5.2 Embedded for Linux/Armv8 64-bit is available within the Product Release Announcement which can be downloaded from https://www.ptc.com/products/developer-tools/apexada .

The addition of the new code generation capability for 64-bit Armv8 processors to ApexAda opens up a whole new landscape for embedded application development using ApexAda. PowerPC processors have for a long time been a design choice for our aerospace and defense customers due to their balance of performance, cost, and power characteristics. Intel processors have offered many of our customers increased performance at a cost of additional complexity and power requirements. Driven by the mobile consumer market, Arm processors provide high performance and low power advantages over Intel processors. We think these advantages combined with the flexibility provided by embedded Linux distributions and the availability of low-cost and high-performance consumer-grade development boards as well as ruggedized 64-bit Arm boards will provide substantial benefits to our customers looking to modernize existing deployed applications while mitigating risks through continued use of the same time-proven and industrial-strength ApexAda compiler technology. The 64-bit Armv8 (aarch64) processors are now well-known and proven processors with a long lifecycle and there are multiple 64-bit Linux distributions available which run on these processors. Follow-on products leveraging the new ApexAda 64-bit Armv8 (aarch64) code generation capability for other real-time operating systems are under development with prioritization based on customer interest and requirements.

If you would like to receive additional information about the new PTC ApexAda v5.2 Embedded for Linux/Intel64 to Linux/Armv8 64-bit product or wish to be contacted by a PTC Developer Tools sales representative regarding evaluations, upgrades and associated pricing, register your request at https://www.ptc.com/en/products/developer-tools/objectada/contact-sales.

Ada and Operating Systems

UNIX OS Written in Ada

From: gdotone@gmail.com
Subject: is there a version of unix written in Ada
Date: Fri, 24 Jul 2020 15:11:47 -0700
Newsgroups: comp.lang.ada

Is there a UNIX-like OS written completely in Ada?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 25 Jul 2020 11:47:35 +0300

The short answer is "no".

There have certainly been operating systems written in Ada -- the OS for the Nokia MPS-10 minicomputer is an example.

There are several real-time kernels and similar low-level SW components written in Ada, but probably they do not qualify as "Unix-like", depending on what you mean by that term.

Why do you ask?

From: Stéphane Rivière <stef@genesix.fr>
Date: Sat, 25 Jul 2020 11:36:57 +0200

See OS section of https://github.com/ohenley/awesome-ada

> There have certainly been operating systems written in Ada -- the OS for the Nokia MPS-10 minicomputer is an example.

Wasn't aware, thanks! Find that... Very few refs on the net...

https://dl.acm.org/doi/abs/10.1145/989798.989799

From: Jesper Quorning <jesper.quorning@gmail.com>
Date: Sat, 25 Jul 2020 07:43:15 -0700

> is there a unix like OS written completely in Ada?

Do not know if it is unix-like, but this [1] looks active. Maybe he needs help..

My own dream was to port GNU/Hurd to Ada while renaming it to something not hurding so much.

[1] https://github.com/ajxs/cxos

From: Andreas Zuercher <ZUERCHER_Andreas@outlook.com>
Date: Sat, 25 Jul 2020 12:20:25 -0700

> is there a unix like OS written completely in Ada?

In 1981, there in fact was one that had 2 public releases with work in progress on Version 3: iMAX-432, depending on how puritanical one wishes to be about what is or is not Unix-like. (iMAX-432 was far more Unix-like than, say, MVS-like or CP/M-like.)

If anyone has an inside negotiating track at Intel (or the contracting firm that Intel hired to develop it), perhaps they would be willing to open-source the old iMAX432 operating system that was released for the iAPX432 processor that was designed from the ground up to have an Ada-centric instruction set. Although it was more Multics-esque than Unix-esque* and although it was written specifically for the iAPX432 (and thus had much iAPX432-only assembly language), it should be relatively easily transliterable into other ISAs because the iAPX432 ISA more closely resembles Java bytecode, LLVM bitcode, and C# CIL/MSIL than other rudimentary machine codes of that era, due to being object-based/OO-lite in the hardware's machine code (which is what doomed the iAPX432 in the early 1980s: it was so complex that it required 3 separate IC dies in 3 separate ceramic packages, and it ran relatively hot).

* Conversely, both Multics & our modern Unix are nowadays birds of the same feather despite the multi-decade dislocation in time from each other, due to both having:

1) multiple threads per address space;
2) multiple DLLs per address-space;
3) multiple memory-mapped files (i.e., mmap(2) in Unixes versus snapping segment-files in Multics);
4) IPC based on multiple threads or multiple processes pending on a single message-queue;

5) soft real-time thread scheduling priorities in addition to time-sharing scheduling priorities;

and

6) a GNU-esque long-form whole-words and short-form abbreviated-letters of each hyphenated command-line flag

are birds of much the same father, as opposed to 1970s-era spartan Unix that abhorred all of these multiplicities, hence AT&T's uni-based name in AT&T's

1970-divorce-from-MIT's/GE's/AT&T's/Honeywell's-Project-MAC in defiance of Project MAC's multi-based name, because the tongue-in-cheek humor of Unix's name as eunuchs is Multics castrated. Eschewing singleton this and singleton that, Unix nowadays is no longer a castrated eunuch, due to reintroducing a cousin-like variant of nearly every multiplicity feature of Multics other than the multiple rings (unless one counts VM hypervisors nowadays as reintroducing a cousin of that one too).

https://en.wikipedia.org/wiki/IMAX_432

From: Stéphane Rivière <stef@genesix.fr>
Date: Sun, 26 Jul 2020 21:45:50 +0200

> I remember someone was writing an OS in Ada, but I do not remember who was, nor the name of the project, nor if it was unix-ish.

In the very old archive https://stef.genesix.org/aide/aide-src-1.04.zip you will find:

- The last RTEMS 3.2.1 Ada sources (yes... old RTEMS releases are offered in two flavors: Ada and C) comes with docs & manuals.
- the Ada sos-os Ada series (based from edu-os in C)

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Date: Mon, 27 Jul 2020 00:15:52 +0200

> - The last RTEMS 3.2.1 Ada sources (yes... old RTEMS releases are offered in two flavors : Ada and C) comes with docs & manuals.

Marte OS implements Minimal Real-Time POSIX.13 in Ada, so it should be Unix-like.

https://marte.unican.es/

The same group recently announced M2OS, which is also in Ada, but not Unix-like.

https://m2os.unican.es/

From: Stéphane Rivière <stef@genesix.fr>
Date: Mon, 27 Jul 2020 09:40:05 +0200

> In 1981, there in fact was one that had 2 public releases with work in progress on Version 3: iMAX-432, depending on how puritanical one wishes to be about what is or is not Unix-like. (iMAX-432 was far more Unix-like than, say, MVS-like or CP/M-like.)

Very interesting Andreas, thanks for this part of Ada and CPU history...

From: Stéphane Rivière <stef@genesix.fr>
Date: Mon, 27 Jul 2020 09:40:04 +0200

> The same group recently announced M2OS, which is also in Ada, but not Unix-like.

> https://m2os.unican.es/

Not aware of that, Thanks Jeffrey

I will test that, the Toolchain is Linux based and includes GDB...

From: nobody in particular <nobody@devnull.org>
Date: Mon, 27 Jul 2020 14:58:36 +0000

> If anyone has an inside negotiating track at Intel (or the contracting firm that Intel hired to develop it), perhaps they would be willing open-source the old iMAX432 operating system that was released for the iAPX432 processor that was designed from the ground up to have an Ada-centric instruction set.

I guess it could be worthwhile contacting Steve Lionel who recently retired from Intel after working for DEC, COMPAQ, HP, on Fortran compilers. He has a blog site, I'll not post the details here so as not to encourage automated spam. Doctor Fortran is his nickname.

Ada on OpenVMS Retake

From: gérard Calliet <gerard.calliet@pia-sofer.fr>
Subject: Ada on OpenVMS, where to have a new beginning
Date: Mon, 17 Aug 2020 19:14:17 +0200
Newsgroups: comp.lang.ada

I participated in a GNAT Ada build for Itanium OpenVMS (https://github.com/AdaLabs/gnat-vms) a few years ago. It is based on a GCC 4.7.3 .

I'm coming back to this work to maintain it and make it evolve, in a general approach of making Ada available in OpenVMS environments (VAX, Alpha, Itanium, and soon x86). (http://www.vmsadaall.org/index.php/en/)

For VAX and Alpha we have at least DEC Ada and Alsys Ada. On Itanium I have to maintain GNAT Ada on GCC. For x86 I have to base on the GNAT Ada front end for LLVM, since VSI ports VMS to x86 (https://vmssoftware.com/updates/state-of-the-port/) basing the compilers on LLVM.

I know that AdaCore dropped commercial support for GNAT Ada on OpenVMS in 2015. It's not the commercial reasons that interest me.

In approaching this project again, I would like to know as much as possible about how far AdaCore's people or helpers have come in their developments for OpenVMS, what problems they have dealt with in the GCC upgrades they have resolved, only considered, and those they have seen as too difficult and blocking. The question arises as well for the upgrades (with for example around this time the transition of the GCC build to C++) as for the evolution of the debug management.

If the answers raise confidentiality issues, I don't want to put anyone in trouble, but I'm looking for indications on who to negotiate with.

It's not impossible that AdaCore's people were among the last to develop GCC for OpenVMS Itanium. They may also be able to inform me about the build of the C and C++ part for GCC OpenVMS. I think indeed to associate to my efforts for Ada the exploration of the availability of a C++ GCC for Itanium OpenVMS.

This resumption of [this] project is quite at its beginning. My goal is to open as much as possible the work and its results to a collaborative work, in Open Source standards. One of my first tasks will be to update the current repository to allow opened development.

From: Andreas Zeurcher <ZUERCHER_Andreas@outlook.com>
Date: Mon, 17 Aug 2020 11:56:56 -0700

For those interested, a hobbyist license of OpenVMS is available from VMS Software, Inc., which is the new owners of VMS instead of HPE. There is also a free Alpha emulator for Windows 10 as well.

https://training.vmssoftware.com/hobbyist

From: nobody in particular <nobody@devnull.org>
Date: Tue, 18 Aug 2020 18:49:13 +0000

It is unlikely yet perhaps Steve Lionel will have some info on this. Although he was not involved with Ada (to my knowledge) he was a fixture in the compiler community for Fortran and probably more, at DEC, COMPAQ, and HP over a long period and might be able to identify likely suspects to contact.

This year the VMS port to Intel X86 was finally completed

https://vmssoftware.com/updates/state-of-the-port/

https://sciinc.com/remotevms/vms_techinfo/vms_news/OpenVMSOnX86-64.asp

I remember a lengthy discussion in the VMS newsgroup many years ago regarding the future of Ada on VMS. I believe the guys at the above companies were involved. I think the conclusion was they would not or could not handle it in-house and I believe the Ada they had on VMS was only 95. There were murmurings that they would try to find somebody to do it but I did not hear that AdaCore ever released anything.

Thank you, I'll follow this thread with interest.

Ada and Other Languages

CLU and Alphard Grammars

From: Oliver Kellogg <olivermkellogg@gmail.com>
Subject: CLU and Alphard grammars available in HTML
Date: Thu, 23 Jul 2020 15:02:09 -0700
Newsgroups: comp.lang.ada

The research languages CLU and Alphard had some influence on the design of Ada [1].

The available grammar documents [2], [3] are in Postscript or PDF format, in the case of Alphard in a somewhat hard to read typeface due to being scanned from the original document.

For an HTML version of the grammars, see

http://okellogg.de/proglang/CLU-syntax.html

http://okellogg.de/proglang/alphard-collected-syntax.html

[1] Ada 83 LRM section 1.3

See e.g. http://archive.adaic.com/standards/83lrm/html/lrm-01-03.html

[2] CLU Reference Manual Appendix A

See e.g. http://okellogg.de/proglang/CLU-syntax.pdf

[3] An informal definition of Alphard

See e.g. http://okellogg.de/proglang/An_informal_definition_of_Alphard.pdf

From: "oliverm...@gmail.com" <olivermkellogg@gmail.com>
Date: Wed, 19 Aug 2020 13:27:33 -0700

Update:

Translation of the full "Informal Definition of Alphard" document to HTML is in progress, see

http://okellogg.de/proglang/an-informal-definition-of-alphard.html

100% completion ETA is within the next few weeks.

Ada Practice

Ada on Apple's New Processors Licensing Concerns

[The thread started with compiler backend concerns, but most of it evolved towards licensing issues in view of the optimizations that the Apple Store may perform to intermediate code. As is often the case with licensing arguments, no entirely satisfactory consensus was reached on the actual situation, and any conclusions in any case should be vetted by qualified experts. One possible takeaway, as Fabien Chateau summarizes in one of his posts, is that the GNAT-LLVM frontend opens many possibilities that did not exist before, which is a net positive in any case.

The complete thread can be found at https://groups.google.com/g/comp.lang.ada/c/lHQQfRATKno --arm]

From: Jerry <list_email@icloud.com>
Subject: Ada on Apple's new processors
Date: Mon, 22 Jun 2020 15:53:00 -0700
Newsgroups: comp.lang.ada

Apple is beginning its third nightmare transition to a new processor family. What does this mean for Ada on macOS?

Can we hope for a native compiler anytime soon? We will have Rosetta 2 until we don't. (Original Rosetta lasted for two OS generations and then it was taken away.) I could tell you the story of needing to run a small PowerPC program to set up a slightly old Apple WiFi device a couple years ago. Buy Parallels. Call Apple and send $30 to get Snow Leopard Server--that's 10.6. Virtualize Snow Leopard Server on Parallels to run the WiFi set-up program in Rosetta.)

From: Vadim Godunko <vgodunko@gmail.com>
Date: Tue, 23 Jun 2020 03:42:46 -0700

> Apple is beginning its third nightmare transition to a new processor family. What does this mean for Ada on macOS?

> Can we hope for a native compiler anytime soon?

I suppose native toolchain will be based on LLVM, thus it will allow to use GNAT LLVM on new processors.

[A large discussion is omitted at this point on the implications of GCC code generation in regard to the Runtime Library Exception (RLE) clause of GPLv3. However, as later was pointed out, GNAT LLVM does not have any relation to GCC.

Arnaud Charlet from AdaCore eventually jumped in to clarify the status of GNAT LLVM licensing, which re-sparked a somewhat more focused discussion in relation to the original topic, which follows. --arm]

From: charlet@adacore.com
Date: Thu, 25 Jun 2020 00:21:01 -0700

> The compiler links to GNAT-LLVM, the runtime doesn't.

> Pretty sure that the AdaCore people said it won't fall under GPL.

That's correct, there is no issue here. The GNAT LLVM compiler is a tool and is licensed under GPLv3, which is just fine and the proper license for a tool. The runtime which is linked with your executable comes from the gcc.gnu.org repository and contains the GCC RunTime exception license.

From: "Luke A. Guest" <laguest@archeia.com>
Date: Thu, 25 Jun 2020 10:55:39 +0100

> That's correct, there is no issue here. [...]

Can you confirm that using FSF GNAT with GNAT-LLVM (GPLv3) does or does not enable the IR clause in the GPLv3?

From: charlet@adacore.com
Date: Thu, 25 Jun 2020 03:14:31 -0700

> Can you confirm that using FSF GNAT with GNAT-LLVM (GPLv3) does or does not enable the IR clause in the GPLv3?

It does not and in any case, invoking this clause is a red herring since as explained in the license, the concern and what's not allowed is using an intermediate representation and feed it to a proprietary (non-GPL-compatible) software to e.g. optimize it or further process it. LLVM is a GPL-compatible Software, so this is irrelevant.

From: Simon Wright <simon@pushface.org>
Date: Thu, 25 Jun 2020 12:03:14 +0100

> It does not and in any case, invoking this clause is a red herring since as explained in the license, the concern and what's not allowed is using an intermediate representation and feed it to a proprietary (non-GPL-compatible) software to e.g. optimize it or further process it.

Optikos has (at last) made clear his concerns about this: if it is indeed the case that Apple requires App Store developers to deliver bitcode for further proprietary optimizations then there might be an issue.

Depends on whether LLVM IR (which I understand is logically equivalent to bitcode) can count as target code? I've seen it described as LLVM assembler ...

From: "Luke A. Guest" <laguest@archeia.com>
Date: Thu, 25 Jun 2020 12:25:59 +0100

> [...] further proprietary optimizations [...] might be an issue.

> Depends on whether LLVM IR [...] can count as target code?

Indeed. The only thing I've found so far is this:

https://thenextweb.com/apple/2015/06/17/apples-biggest-developer-news-at-wwdc-that-nobodys-talking-about-bitcode/

Quote from near the top:

'This means that apps can automatically "take advantage of new processor capabilities we might be adding in the future, without you re-submitting to the store."'

From the apple docs it links to at the top:

"Bitcode is an intermediate representation of a compiled program. Apps you upload to App Store Connect that contain bitcode will be compiled and linked on the App Store. Including bitcode will allow Apple to re-optimize your app binary in the future without the need to submit a new version of your app to the App Store. "

So, it looks like he [Andreas Zuercher, aka Optikos --arm] is right.

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Tue, 30 Jun 2020 04:16:38 -0700

> this later closed-source processing of the app by Apple for nonjailbroken ARM-based Macs and iDevices to distribute via the App Store seems to violate terms of at least the RLE [Runtime Library Exception] if not GPLv3 too.

The Apple app store is incompatible with the GPL since long ago: https://www.fsf.org/blogs/licensing/more-about-the-app-store-gpl-enforcement

I don't see anything new here.

From: Optikos <ZUERCHER_Andreas@outlook.com>
Date: Tue, 30 Jun 2020 05:28:45 -0700

>

> The Apple app store is incompatible with the GPL since long ago: https://www.fsf.org/blogs/licensing/more-about-the-app-store-gpl-enforcement

Yes, when the developer's app is GPLed, the App Store's terms and the GPL's terms are mutually incompatible. Historically, GPLing an app would have been by developer choice (unless somehow violating the RLE which was rare in practice because using garden-variety unmodified IR-unadorned GNAT, GCC, and so forth resulted in an Eligible Compilation Process in RLE).

> I don't see anything new here.

What is new here is that there appear to be well-reasoned ways (e.g., the Wide legal theory along this thread [that LLVM IR is a kind of IR code according to the RLE --arm]) that GNAT-LLVM could •force• a developer's app to [be] GPLed against the developer's will by easy-to-enact-in-GNAT-LLVM violations of the RLE's terms that cause the Compilation Process to not achieve the stricter Eligible Compilation Process definition, due to Apple's closed-source manipulations of LLVM IR bitcode.

Perhaps the work-around is that GNAT-LLVM-based developers of apps should •never• submit LLVM IR bitcode to Apple's App Store's app-intake procedure. In the past as far back as 2015, submitting bitcode instead of machine code was optional. It is unclear with the new ARM-based Macs, whether that optionality will continue in the future, or whether that optionality has already been curtailed.

(Conversely, under the Narrow legal theory along this thread [that LLVM IR is equivalent to assembly code and not an actual IR for RLE purposes --arm], your claim is correct, nothing has changed: if an app-developer doesn't want to suffer the mutual incompatibility of the GPL and Apple App Store, then don't choose GPL as the license for the app, because despite its name LLVM IR bitcode is merely assembly language which is unregulated by RLE.)

From: charlet@adacore.com
Date: Tue, 30 Jun 2020 07:35:03 -0700

> We need clarification on whether the translation from GCC's IR to LLVM's IR invokes this clause. I'm not sure if GNAT final IR before the GNAT-LLVM backend is GENERIC or GIMPLE.

GNAT LLVM doesn't use nor depend on GCC at all: it goes directly from the GNAT tree to LLVM bitcode, there is never any GENERIC nor GIMPLE in sight by design and never can be (unlike with the old DraggonEgg FWIW).

> I've had a quick look in GNAT-LLVM and I cannot see any flags enabling the output of GCC's IR, only LLVM's IR.

See above.

By the way the reason I haven't answered other messages is mainly because I am not familiar with Apple's specific constraints here, so I'd rather not make any statement about them rather than making wrong statements and you shouldn't draw any conclusion from the fact that I haven't replied to some of the messages in this thread.

From: "Luke A. Guest" <laguest@archeia.com>
Date: Tue, 30 Jun 2020 15:46:43 +0100

> GNAT LLVM doesn't use nor depend on GCC at all: it goes directly from the GNAT

Ok, makes sense.

> tree to LLVM bitcode, there is never any GENERIC nor GIMPLE in sight by design and never can be (unlike with the old DraggonEgg FWIW).

But, GNAT is 1 part of GCC and the GPLv3 mentions IR, what constitutes the IR? Surely it covers the Ada AST IR?

Does the GPL infect across the different IR boundaries?

[...]

From: Simon Wright <simon@pushface.org>
Date: Tue, 30 Jun 2020 21:01:59 +0100

> GNAT LLVM doesn't use nor depend on GCC at all: it goes directly from the GNAT tree to LLVM bitcode, there is never any GENERIC nor GIMPLE in sight by design and never can be (unlike with the old DraggonEgg FWIW).

It seems to me that there's a lot of argument about things which can't or won't be changed.

AdaCore have produced GNAT-LLVM as a proof of concept, aimed really at targets not supported by GCC but of interest to AdaCore's customers.

GNAT-LLVM code itself is (C) AdaCore, and is GPLv3. The gcc/ada code is (C) FSF, and is GPLv3.

No change there.

The current build takes the RTS from FSF GCC, though clearly it could take it from elsewhere (e.g. some bare metal RTS).

That RTS is (C) FSF, GPLv3 + runtime exception.

Some here have thought, Aha! LLVM, RTS with runtime exception, people could produce apps for iOS!!!!

Then, cold reality strikes: it looks as though there's a conflict between the actual terms of the runtime exception and Apple's requirements for code to be submitted to the App Store (it needs to be in LLVM IR or equivalent); the code would very likely lose the protection of the runtime license umbrella.

Now, guys, given that there's Apple on one side standing on a mountain of money and a prickly attitude to what they'll accept for their app store, and on the other side a very much smaller developer community, who's going to risk going to court to put a GNAT app on to the App Store?

Whether you could make such an app and run it on iPhones privately, without going through Apple & the App Store, I don't know. I'm sure the NSA can.

[...]

From: "Luke A. Guest"
<laguest@archeia.com>
Date: Tue, 30 Jun 2020 21:20:27 +0100

>> By that point, there should be a strong track record of technical knowledge regarding Apple's bitcode submission policies to the App Store to relay to the attorneys so that they can simply turn the legal crank to make a decision/adjustments of whether/how GNAT-LLVM is to transition out of experimental status.

> I'd have thought that AdaCore's response to this idea would be to ask where you got the idea that iOS & the App Store would feature as a candidate target.

I'm developing SDLAda, there are mobile targets. I don't see why Ada shouldn't. Jesus, even COBOL can compile to mobile according to an article I read a while ago. If AdaCore and Ada users want people not thinking that Ada is an ancient language, then it needs to wake up, smell the coffee and get on mobile.

From: Wesley Pan
<wesley.y.pan@gmail.com>
Date: Tue, 30 Jun 2020 15:07:05 -0700

> I'm developing SDLAda, there are mobile targets. I don't see why Ada shouldn't. [...]

I COMPLETELY agree with Luke! We need Ada to expand to things like mobile, gaming, and other "more exciting" markets to help attract the new generations of software engineers and to stay relevant in the public's eyes. The gaming industry alone rivals that of Hollywood. By the end of 2019, GTA 5 sold more than 100 million copies worldwide, earning its publisher more than $6 billion on a $265 million development budget. That's not chump change. How can members of the Ada community ever really jump into such industries if the same issues like the license keep coming up as roadblocks?!

I'm not in any way suggesting the Ada community give up its focus on the safety and reliability angle. Those are very important too. But, if you were to have affordable/free Ada tools for mobile/gaming on one side, and expensive tools for the next-gen Mars rover on the other side, which do you think would attract more end users?

BTW, even the new "cool" Rust language is being used to develop apps for mobile. Ada apps....?

From: Optikos
<ZUERCHER_Andreas@outlook.com>
Date: Tue, 30 Jun 2020 21:46:47 -0700

> I'd have thought that AdaCore's response to this idea would be to ask where you got the idea that iOS & the App Store would feature as a candidate target.

Gee, either

a) we all concurrently pulled it out of thin air via overactive imagination as you imply,

or

b) the following extant events & facts transpired:

At least an engineer at AdaCore (if not AdaCore speaking as an organization) wrote the following on the GNAT-LLVM repository's README.md:

“[GNAT-LLVM] is a work-in-progress research project that's not meant for and shouldn't be used for industrial purposes. It's meant to show the feasibility of generating LLVM ••bitcode•• for Ada.” (emphasis added)

LLVM.org did not organically produce bitcode out of their own volition. Bitcode was Apple's idea, Apple's design, contributed by Apple to benefit primarily Apple as a strategic technology to facilitate Apple's OS-optimization & processor-switcheroo goals without Apple begging all app developers to resubmit a plethora of minor-variation apps every time Apple has a bright idea or Big New Thing. So when GNAT-LLVM's README.md is explicitly calling out bitcode emission as the A#1 top-priority reason for GNAT-LLVM to exist, by using that very term bitcode, it is quite clear that the intended reading of README.md is referring to the Apple-Apple-Appleness of bitcode since bitcode was announced at Apple's Worldwide Developer Conference in June 2015 as a key technology related to App Store submission and downstream proprietary processing by Apple post-submission:

https://TheNextWeb.com/apple/2015/06/17/apples-biggest-developer-news-at-wwdc-that-nobodys-talking-about-bitcode

From: Fabien Chouteau
<fabien.chouteau@gmail.com>
Date: Wed, 1 Jul 2020 02:23:29 -0700

> Gee, either

> a) we all concurrently pulled it out of thin air via overactive imagination as you imply,

> or

> b) the following extant events & facts transpired:

The answer is a).

> it is quite clear that the intended reading of README.md is referring to the Apple-Apple-Appleness of bitcode since bitcode was announced at Apple's Worldwide Developer Conference in June 2015 as a key technology related to App Store submission and downstream proprietary processing by Apple post-submission:

It seems like you focus too much on details of a simple README. LLVM bitcode is sometimes used to talk about the general LLVM IR.

The example use cases mentioned by the README are bringing more tooling to the Ada ecosystem, for instance with KLEE, or "connecting the GNAT front-end to the LLVM code generator".

It took time and effort to publish GNAT-LLVM on GitHub, and AdaCore had absolutely no obligation to do so. To be honest, I am personally a bit disappointed to see such a long discussion on what is allegedly not possible to do with GNAT-LLVM (and was absolutely not possible before anyway), rather than all the possibilities that GNAT-LLVM opens.

From: Simon Wright
<simon@pushface.org>
Date: Wed, 01 Jul 2020 12:03:10 +0100

> It took time and effort to publish GNAT-LLVM on GitHub, and AdaCore had absolutely no obligation to do so. To be honest, I am personally a bit disappointed to see such a long discussion on what is allegedly not possible to do with GNAT-LLVM (and was absolutely not possible before anyway), rather than all the possibilities that GNAT-LLVM opens.

Personally, I thank AdaCore for making such an interesting project available.

A couple of postings down Jeffrey Carter quoted this, which seems apt in the current context:

"Propose to an Englishman any principle, or any instrument, however admirable, and you will observe that the whole effort of the English mind is directed to find a difficulty, a defect, or an impossibility in it. If you speak to him of a machine for peeling a potato, he will pronounce it impossible: if you peel a potato with it before his eyes, he will declare it useless, because it will not slice a pineapple."

Charles Babbage

From: Wesley Pan
<wesley.y.pan@gmail.com>
Date: Thu, 2 Jul 2020 17:51:36 -0700

> It took time and effort to publish GNAT-LLVM on GitHub [...]

Hi Fabien,

That's a fair point. As with any compiler related development (and software in general), I'm sure the amount of time and effort it took to create GNAT-LLVM was significant. Aside from the licensing issue/debate, it is a really great contribution to the Ada community and I hope it becomes production quality in the very near future.

AdaCore is the main (if not only) company that continues to make innovative and very helpful tools related to Ada (e.g. libadalang and LearnAda). As you pointed out, AdaCore was not obligated to make such contributions. GNAT-LLVM could very well have been kept in closed doors to only further AdaCore's internal development.

When news about GNAT-LLVM first came out, I for one thought it would finally allow people to create IOS apps in Ada and to further the adoption of the language. Sadly, not sure that will ever happen now...

From: gautier_niouzes@hotmail.com
Date: Fri, 3 Jul 2020 04:08:00 -0700

> It took time and effort to publish GNAT-LLVM on GitHub [...]

There is a bias here: the people discussing on comp.lang.ada tend to be busy... discussing on comp.lang.ada - and less busy doing actual programming. Chatting and programming are incompatible activities IMHO. At least you cannot do both at exactly the same time...

From: Simon Wright <simon@pushface.org>
Date: Thu, 02 Jul 2020 10:54:27 +0100

> "The LLVM code representation is designed to be used in three different forms: as an in-memory compiler IR, as an on-disk bitcode representation (suitable for fast loading by a Just-In-Time compiler), and as a human readable assembly language representation", which to me precisely matches "data in any format that is used as a compiler intermediate representation, or used for producing a compiler intermediate representation".

On thinking about this further, I can't help wondering whether this is deliberate.

From: antispam@math.uni.wroc.pl
Date: Fri, 3 Jul 2020 17:18:40 +0000

> On thinking about this further, can't help wondering whether this is deliberate.

Why doubt? FSF clearly did not want what Apple is doing now. Apple understood this well, left GCC development and started promoting LLVM. FSF lawyers formulated appropriate licencing language. So the remaining question is if they did a good job. Basically folks here are searching for a loophole. Loopholes happen, but FSF was careful, so do not bet on this.

From: Optikos <ZUERCHER_Andreas@outlook.com>
Date: Fri, 3 Jul 2020 11:31:33 -0700

> Why doubt? [...]

These have been my exact concurring conclusions as well for over 2 years now, when I back then ceased coding up my own variant resembling what is now known as GNAT-LLVM. Some of my design/coding work was hinted at in multiple of my postings here on c.l.a. back then. I figured out these ••chilling effects•• on my own over 2 years ago.

Question about Best Practices with Numerical Functions

From: mockturtle <framefritti@gmail.com>
Subject: Question about best practices with numerical functions
Date: Fri, 3 Jul 2020 22:30:52 -0700
Newsgroups: comp.lang.ada

I have a question about the best way to manage a potential loss of precision in a numerical function. This is a doubt that came to my mind while writing a piece of software; now I solved the specific problem, but the curiosity remains.

Let me explain.

Recently I needed to write an implementation of the Lambert W function (is the function that given y finds x such that x*exp(x)=y). This function cannot be expressed with elementary functions and the algorithm I found basically solves the equation in an iterative way. Of course, if you fix the maximum number of iterations, it can happen that the convergence is not fast enough and you obtain a result that is potentially less precise than what you would expect.

I was wondering how to manage such a non convergence case. Please note that I am supposing that I am writing a "general" function that could be used in many different programs. If the function was specific for a single program, then I would choose the line of action (e.g., ignore, log a warning or raise an exception) depending on the needs of the specific program.

(Incidentally, it turned out that the implementation converges nicely for any value of interest; nevertheless, the curiosity remains...)

I can see few line of actions that would make sense

[1] Raise an exception.

Maybe this is a bit too drastic since there are cases where a moderate loss of precision does not matter (this was my case, i just needed one or two decimal digits)

[2] Let the function have an optional "precision" parameter and raise an exception if the precision goes below that

[3] Let the function return a record with a field Value with the actual result and a field Error with the estimated precision.

This would make the code a bit heavier since instead of calling

```
X := Lambert(Y);
```

you would say

```
X := Lambert(Y).Value;
```

Not really a huge deal, however...

[4] Print a warning message to standard error or some logging system and go on.

This sounds like the worst option to me. The message could be overlooked and, moreover, it supposes there is some logging facilities or that the standard error is available for logging... Remember that the function should be general, to be used in any program.

[5] ???

Any suggestions?

Thank you in advance

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Sat, 4 Jul 2020 09:50:57 +0200

> [5] ???

[5] Interval computations is the best way to handle rounding errors:

https://en.wikipedia.org/wiki/Interval_arithmetic

An Ada implementation is here:

http://www.dmitry-kazakov.de/ada/intervals.htm

From: "Nasser M. Abbasi" <nma@12000.org>
Date: Sat, 4 Jul 2020 05:45:57 -0500

Most Fortran Lapack use INFO code.

"All documented routines have a diagnostic argument INFO that indicates the success or failure of the computation, as follows:

INFO = 0: successful termination

INFO < 0: illegal value of one or more arguments -- no computation performed

INFO > 0: failure in the course of computation"

https://www.netlib.org/lapack/lug/node138.html

So you could follow that.

[...]

From: Shark8 <onewingedshark@gmail.com>
Date: Mon, 6 Jul 2020 10:02:39 -0700

> [5] ???

> Any suggestions?

One way to do this would be to use fixed-point types:

(1) Convert your input to the fixed-point that has a "good enough" delta for the precision you want.

(2) Run the algorithm.

(3) Convert back to your normal value-type.

This assumes you're using floating-point or integers, but one nice thing about fixed-point is that it has a bounded error when dealing with operations, unlike floating-point. -- I remember some years ago seeing a bug report dealing with floating-point, where the particular error simply couldn't have happened with fixed-point.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Tue, 7 Jul 2020 00:23:06 +0200

> One way to do this would be to use fixed-point types:

Rounding error is bounded in both cases. Fixed-point has same error regardless of the values involved in the operations. Floating-point has error depending on the values.

I would say that floating-point error would be roughly the same for addition, subtraction and multiplication, provided fixed-point does not overflow. It will be hugely better for division.

Using fixed-point arithmetic has only sense for a few marginal cases of rounding.

Furthermore converting many algorithms to fixed-point might turn quite non-trivial as you will have to ensure absence of overflows and underflows. Where floating-point computation just would lose some precision, fixed-point will catastrophically fail.

General Circular Buffer

From: Daniel <danielnorberto@gmail.com>
Subject: General circular buffer example not tied to any specific type
Date: Sat, 4 Jul 2020 10:00:26 -0700
Newsgroups: comp.lang.ada

Hello, any theoretical example of buffer I can find is always tied to a specific type.

I'm looking for any example of Ravenscar buffer able to use any type of data at the same time.

I suppose it will need to serialize all data and manipulate it as a group of bytes.

Does anybody know any example of this written in Ada?

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Sat, 4 Jul 2020 19:25:22 +0200

Ring buffer of indefinite elements would be OK. As an element you can use this:

```
type Item (Size : Stream_Element_Count)
is record
   Data : Stream_Element_Array (1..Size);
end record;
```

Instantiate the generic buffer with this type. Use stream attributes to serialize/deserialize.

Alternatively you can do it with Storage_Element in the above and use a fake storage pool to store/restore objects. Or a combination "for X'Address use Y" with pragma Import (Ada, X);

If the type set is somewhat statically known you can use a variant record as an element too.

In some cases you can have a ring buffer of type tags and a set of ring buffers. For each type tag you would keep values in a separate ring buffer.

From: "J-P. Rosen" <rosen@adalog.fr>
Date: Thu, 24 Sep 2020 06:39:43 +0200

> Does anybody knows any example of this written in Ada?

Hmmm, you know, Ada is a strongly typed language, therefore what you put in a buffer must have a well defined type.

There are two possibilities:

1) If you can accept several buffers, one for each type, make it generic and instantiate it as many times as you need

2) Make a buffer of Stream_Elements, and use the streaming attributes ('Read, 'Write) to turn any type into stream elements.

Ada.Streams.Stream_IO can also be handy in some cases.

From: Simon Wright <simon@pushface.org>
Date: Fri, 25 Sep 2020 15:32:01 +0100

As J-P has said, you could use 'Write and 'Read (or better, 'Output and 'Input) to write to a stream.

The beginnings of an alternative, which I last worked on a while ago, is at [1]; it's an Ada implementation of part of MessagePack[2] (boolean, integer, float, string). Still a way to go!

Writing arbitrary data to a stream using 'Write/'Output suffers from the disadvantage that the reading side won't know what to expect unless you have some protocol in place. This Message_Pack doesn't eliminate this at all.

For a while, I supported a scheme where all the data to be transmitted had to be instances of a tagged type e.g. Base; as far as I can remember, you output the data using Base'Class'Output and read it in using Base'Class'Input.

[1] https://sourceforge.net/u/simonjwright/msgpack-ada/code/ci/master/tree/

[2] https://en.wikipedia.org/wiki/MessagePack

Fixed vs Float Precision and Conversions

From: Björn Lundin <b.f.lundin@gmail.com>
Subject: Fixed vs float and precision and conversions
Date: Tue, 7 Jul 2020 23:10:20 +0200
Newsgroups: comp.lang.ada

I've for years run an interface towards external part on a raspberry pi that communicates with JSON over http (JSONRPC2)

the versions are [...]

I have found this reliable but suddenly I have got some rounding troubles. Or I perhaps just discovered it now.

I have a fixed type

```
type Fixed_Type is delta 0.001 digits 18;
```

but JSON does not support that. So I get floats instead. I use gnatcoll.json as parser by the way

[Skipped example that boils down to converting a float to a fixed point type. --arm]

The message (JSON) contains a value 5.10 (in a float), but that is converted (sometimes I think) to the fixed_type variable with value 5.099. This gets me into trouble further down in the code.

So - What should I do instead?

should I express my Fixed_Type in another way?

I need to be able to express 6.5 % (0.065) which I could not with type Fixed_Type is delta 0.01 digits 18;

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 8 Jul 2020 00:30:29 +0300

According to RM 4.6(31), conversion to a decimal fixed-point type does not round, but truncates toward zero if the operand is not a multiple of the "small" of the target type, which is usually the case here if Floats are base-two.

You should perhaps change the conversion (Target := Fixed_Type(Tmp)) to round, by doing Target := Fixed_Point'Round (Tmp).

Note, I haven't tried it.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Wed, 8 Jul 2020 20:10:35 +0200

The main reason for asking was to see if I got the whole concept of fixed types wrong or not.

I did expect 'You should do this or that one-liner' as Niklas proposed. I did not get that to work though.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 8 Jul 2020 21:21:45 +0300

> I did expect 'You should do this or that one-liner' as Niklas proposed. I did not get that to work though

Oh. What happened when you tried? How did it fail?

From: Shark8 <onewingedshark@gmail.com>
Date: Tue, 7 Jul 2020 14:58:40 -0700

> but JSON does not support that. So I get floats instead.

This is a limitation of JSON, IIUC: all numeric are IEE754 floats -- see: https://www.json.org/json-en.html

If you have access to both sides of the serialization, you could define an intermediate serialization say a string of "Fixed_Type#value#" where 'value' is the string-representation you need. -- You can extract the value by indexing on the '#' characters, extracting the portion in between, and feeding that via Fixed_Type'Value(EXTRACTED _SUBSTRING), and produce it via "Fixed_Type#" & Fixed_Type'Image(fp_value) & '#'.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Wed, 8 Jul 2020 20:36:47 +0200

> The main reason for asking was to see if I got the whole concept of fixed types wrong or not.

Fixed-point is conceptually a scaled integer. You should deal with it accordingly. [It could be a bit surprising in Ada where conversion to integer rounds. In most languages conversion to integer truncates]

[...]

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Wed, 8 Jul 2020 21:39:20 +0200

> Oh. What happened when you tried? How did it fail?

I did a test routne like below, but realized that Float(5.10) - which was converted to Fixed_Type(5.099) is a valid fixed_type of course.

so

```
Fix1 := Fixed_Type(Flt);
```

or

```
Fix2 := Fixed_Type'Round(Flt);
```

does not really matter, since both may return 5.099 when given 5.10

[...]

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 8 Jul 2020 23:34:46 +0300

> I did a test routine like below, but realized that Float(5.10) - which was converted to Fixed_Type(5.099) is a valid fixed_type of course.

No, see below. You are confusing decimal (base-10) reals with binary (base-2) floats. [...] The point is that Float'(5.10) is not exactly 5.10, because base-2 floats cannot represent decimal fractions exactly. Since the result, as you showed in your first post, of converting (with truncation) Float'(5.10) to Fixed_Type is 5.099, the actual (binary) value of Float'(5.10) is a little less than 5.10, so the truncation gives 5.099 instead of 5.100.

But Fixed_Type'Round (Float'(5.10)) will always give 5.100.

[...]

Binary Search SPARK Proof

From: mockturtle <framefritti@gmail.com>
Subject: My new post on dev.to about SPARK
Date: Thu, 9 Jul 2020 07:16:20 -0700
Newsgroups: comp.lang.ada

first a bit of disclaimer: this is about a recent post of mine on dev.to I post this here since I think that maybe someone in this group could be interested.

Recently I wrote a small binary search procedure for a software of mine. Since I always wanted to start using SPARK, I thought that this could be a nice small problem to start playing around with SPARK. The post on dev.to is about my experience.

If you are curious

https://dev.to/pinotattari/proving-the-correctness-of-a-binary-search-procedure-with-spark-ada-34id

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Thu, 9 Jul 2020 08:27:16 -0700

I shared it on reddit: https://www.reddit.com/r/programming/comments/ho4zzp/proving_the_correctness_of_a_binary_search/

Go upvote :)

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Date: Thu, 9 Jul 2020 18:35:39 +0200

> Recently I wrote a small binary search procedure for a software of mine.

This is good, but why did you write it from scratch? Why not start with an available, reusable binary search? Then you would have a proven, generally useful component.

This is an interesting pedagogical example, but the actual algorithm is too specialized to be of general use.

From: Simon Wright <simon@pushface.org>
Date: Thu, 09 Jul 2020 21:00:18 +0100

Interesting!

I thought to have a bit of a play with it, and I found that neither CE 2019 nor CE 2020 will prove as is, including "assertion might fail, cannot prove Bottom < Top"; but it proves just fine with

```
type Element_Type is new Integer;
```

or

```
subtype Element_Type is Integer;
```

From: "J-P. Rosen" <rosen@adalog.fr>
Date: Fri, 10 Jul 2020 06:17:21 +0200

Hmmm... The following O(N**2) function:

```
function Is_Sorted (Table : Array_Type)
return Boolean
  is (for all L in Table'Range =>
       (for all M in Table'Range =>
          (if L > M then Table (L) > Table (M))))
  with Ghost;
```

can be changed to a O(N) function:

```
function Is_Sorted (Table : Array_Type)
return Boolean
  is (for all L in Table'First .. Table'Last -1
   => Table (L) < Table (L+1))
  with Ghost;
```

From: Paul Rubin <no.email@nospam.invalid>
Date: Thu, 09 Jul 2020 23:04:03 -0700

> Hmmm.. The following O(N**2) function: [...] can be changed to a O(N) function [...]

Should it matter? The code is never executed. It's only used as a specification for the theorem prover.

By the way, Riccardo, thanks for posting that. It was impressive to see that an executable-looking spec like that could be proved automatically with the help of just a few pragmas. I hadn't posted yet because I haven't yet had a chance to try building and playing with the program.

From: "J-P. Rosen" <rosen@adalog.fr>
Date: Fri, 10 Jul 2020 09:47:16 +0200

> Should it matter? The code is never executed.

For Spark, no (although I think that the simpler version is more understandable). But if you run it through an Ada compiler with assertions on, then it will make a difference.

One Discriminated Task per CPU

From: Olivier Henley <olivier.henley@gmail.com>
Subject: 'Number_Of_CPUs' tasks creation, with discriminants, running simultaneously.
Date: Mon, 20 Jul 2020 06:51:12 -0700
Newsgroups: comp.lang.ada

My goal is to distribute similar work across multiple tasks. Incidentally, I want those tasks to start simultaneously, each task having some 'indexed' work (discriminants), and ideally block from main until they are done with their workload.

I got this working by declaring the tasks individually. What I would like to achieve is to leverage 'System.Multiprocessors. Number_Of_CPUs' for the number of tasks created. What is the idiomatic way of achieving what I want?

I tried with an array of tasks, but the problem becomes I do not know either how to start them simultaneously with parameterization or coordinate their exit point with main.

I am lurking for the most 'clean/simple' solution possible.

You can see the actual working code fixed at 8 tasks here: https://github.com/ohenley/xph_covid19/blob/master/src/xph_covid19.adb#L280-L287

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Date: Mon, 20 Jul 2020 19:45:54 +0200

For simple parameterization, you can use a discriminant with a default that is a function call:

```
subtype Task_ID is Integer range
   0 .. System.Multiprocessors.
   Number_Of_CPUs;
subtype Valid_Task_ID is Task_ID range
   1 .. Task_ID'Last;
Last : Task_ID := 0;
function Next_ID return Valid_Task_ID is
begin
   Last := Last + 1;
   return Last;
end Next_ID;
task type T (ID : Valid_Task_ID := Next);
type T_Set is array (Valid_Task_ID) of T;
Worker : T_Set;
```

Each Worker (i) will have its ID determined during elaboration, and they will all start at the "begin" that follows the declaration of Worker. The order of the discriminants is arbitrary; there is no guarantee that Worker (I) will have ID of I. Elaboration is sequential, so each Worker will have a unique ID.

(I think Ada 2X will allow a way to insure that the ID equals the index, but I'm not sure how it will work.)

To block a subprogram until the Worker tasks all complete, declare them in a block statement:

```
Create_Workers : declare
   Worker : T_Set;
begin
   null;
end Create_Workers;
```

From: Olivier Henley <olivier.henley@gmail.com>
Date: Mon, 20 Jul 2020 13:31:45 -0700

> The order of the discriminants is arbitrary; there is no guarantee that Worker (I) will have ID of I. Elaboration is sequential, so each Worker will have a unique ID.

As long as all the tasks get a unique ID, I am fine.

Function as a discriminant at elaboration ... should have thought about it but it looks like I am missing some wisdom points.

Thank you Jeffrey.

From: onox <denkpadje@gmail.com>
Date: Wed, 22 Jul 2020 12:05:01 -0700

Another way is to use an extended return:

spec:

```
type Worker;

task type Worker_Task (Data : not null
access constant Worker);

type Worker is limited record
   ID : Positive;
   T  : Worker_Task (Worker'Access);
end record;

type Worker_Array is array (Positive
range <>) of Worker;

function Make_Workers return
Worker_Array;
body:
function Make_Workers return
Worker_Array is
begin
   return Result : Worker_Array
      (1 .. Positive (Count)) do
      for Index in Result'Range loop
         Result (Index).ID := Index;
      end loop;
   end return;
end Make_Workers;

Workers : constant Worker_Array :=
   Make_Workers;
pragma Unreferenced (Workers);
```

Two Ada 2012 Vendors

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Subject: Two Ada-12 Vendors
Date: Thu, 23 Jul 2020 14:13:35 +0200
Newsgroups: comp.lang.ada

It appears that PTC ObjectAda 10.x is an Ada-12 compiler, making two vendors with Ada-12 compilers*. Only took 8 years.

https://www.ptc.com/-/media/Files/PDFs/Developer-Tools/PTC-ObjectAda-64-for-Windows-10_1_RB.pdf

*An Ada-12 compiler implements at least the entire core language of ISO/IEC 8652:2012. Please don't clutter this thread with posts about compilers that don't meet this definition.

From: Dirk Craynest <dirk@orka.cs.kuleuven.be>
Date: Thu, 23 Jul 2020 17:24:39 -0000

>https://www.ptc.com/-/media/Files/PDFs/Developer-Tools/PTC-ObjectAda-64-for-Windows-10_1_RB.pdf

The above PDF is an announcement from May 27, 2019, and mentioned that the "release expands the support for Ada 2012 language features to include the complete set of Ada 2012 container packages and support for the associated Ada 2012 language constructs required by those packages". Hence a partial Ada 2012 implementation.

But perhaps even more interesting, PTC announced yesterday, July 22, 2020, the release of PTC ObjectAda for Windows Version 10.2: https://developer-tools-us.ptc.com/Announcements/Products/ObjectAda/1000/10.2/RB-20200722-ObjectAda%20for%20Windows%20V10.2.pdf

And the subtitle of that announcement reads: "New native Ada compiler release provides complete Ada 2012 language support".

A small extract from the text:

<start_quote>

"ObjectAda for Windows version 10.2 represents the completion of the phased implementation strategy PTC adopted for Ada 2012 language feature support within the ObjectAda technology.", stated Shawn Fanning, Software Development Director at PTC. "With ObjectAda for Windows version 10.2, the ObjectAda compiler conforms to the Ada Conformity Assessment Test Suite (ACATS) version 4.1Q and adds several new features including support for storage subpools and the Default_Storage_Pool pragma, execution time enforcement of type invariants, and complete support for new Ada expression forms.

<end_quote>

For more information, see the PDF at the 2nd URL above.

Dirk

Dirk.Craeynest@cs.kuleuven.be (for Ada-Belgium/Ada-Europe/SIGAda/WG9)

Survey on the Future of GNAT Community Edition

From: Wesley Pan <wesley.y.pan@gmail.com>
Subject: AdaCore's survey regarding the future of GNAT Community Edition
Date: Fri, 24 Jul 2020 14:42:16 -0700
Newsgroups: comp.lang.ada

I just discovered this via Ada Planet...

Link to the Google Docs survey: https://docs.google.com/forms/d/e/1FAIpQLSet9x3UNUFmfWt5v-8Jb7dW8BgKiJxyEMJ_TFm0G2UJKx5OmQ/viewform

Reddit discussion: https://www.reddit.com/r/ada/comments/hwgbwa/survey_on_the_future_of_gnat_community/

Survey summary reproduced below...

GNAT Ecosystem Community Survey

Hello Ada supporters,

We are writing this message here to present, discuss and get feedback on a plan that we at AdaCore want to put in place. Over the next couple of years, we want to experiment with an evolution of the GNAT ecosystem and would like your help.

So far, there are three grand families of GNAT releases:

- GNAT Pro: An AdaCore release with professional support and high level quality assurance. Available on many different targets (PowerPC, Leon, vxWorks, etc.).
- GNAT Community: An AdaCore release with a lower level of quality assurance, less targets, and a pure GPL license for the run-time.
- GNAT FSF: community built compiler from the FSF source tree. Available from Linux distributions or Msys2 on Windows, for instance.

Moving forward, we are looking to simplify the situation and remove GNAT Community from the picture.

The plan is to reach a point where AdaCore would not release GNAT Community compilers and instead instruct non-professional users to use GNAT FSF builds. We would still keep making GNAT Studio and SPARK releases, and libraries such as AWS and xmlada will be available in the Alire package manager (http://alire.ada.dev). With this plan we also want to invest some more time to help the maintainers of GNAT packages in Linux, BSD, or Windows (msys2) distributions, for instance, and potentially contribute when necessary. Our intention is to contribute to various communities building GNAT packages so that what can be done today with GNAT Community will be doable tomorrow from these community-led builds.

Why are we working on this plan?

We have noticed that GNAT Community's pure GPL license on the run-time is seen as a barrier to new Ada users. More specifically, understanding the consequences of the GPL licence is complex. The result is that newcomers will often be introduced to Ada/SPARK by a legal licence discussion rather than looking at the value of the technology. This will, understandably, scare people off.

On top of this, we are witnessing a widespread misunderstanding around the openness of the Ada language and the GNAT compiler, some people seem to think that Ada and GNAT are proprietary technologies. We see this phenomenon as detrimental to the growth of the Ada community. Of course this misunderstanding will not fade in a couple days, but we think that removing GNAT Community will make the situation clearer and will allow us to better communicate on the situation of the Ada compiler ecosystem.

Besides general comments and discussion around this plan, we would appreciate your feedback in this survey form. Please help us spread the word. The more feedback we get, the more we will be able to move in the right direction.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 24 Jul 2020 19:17:30 -0700

Interesting that they did _not_ post about this survey in this newsgroup. I guess we're just _so_ yesterday ...

From: Stéphane Rivière <stef@genesix.fr>
Date: Sat, 25 Jul 2020 12:35:39 +0200

Many thanks Wesley!!!

Fascinating... (C) Spock

Finally, we're back to the previous situation without the GPL barrier...

GNAT 3.15p was born again ;)

All the arguments given are exactly those I addressed to AdaCore at the highest level at the time (may be about 15 years ago?)...

Originally, GNAT was funded precisely to be available to everyone, including commercial use.

I answered their survey, in a constructive way, of course.

Very good news.

From: Stéphane Rivière <stef@genesix.fr>
Date: Sat, 25 Jul 2020 12:35:43 +0200

> Interesting that they did _not_ post about this survey in this newsgroup. I guess we're just _so_ yesterday ...

I deeply agree! NGs are so (too?) efficient.

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Mon, 27 Jul 2020 01:36:37 -0700

> Interesting that they did _not_ post about this survey in this newsgroup. I guess we're just _so_ yesterday ...

I was going to do it this week ;)

From: DrPi <314@drpi.fr>
Date: Mon, 27 Jul 2020 10:07:48 +0200

What about Ada for microcontrollers?

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Mon, 27 Jul 2020 01:38:35 -0700

> What about Ada for microcontrollers?

With this plan arm-elf and riscv-elf toolchain will be available for Linux and Windows at least.

I am doing a lot of microcontroller programming myself so don't worry about that :)

From: foo wong <crap@spellingbeewinnars.org>
Date: Wed, 29 Jul 2020 02:44:08 -0700

[...]

I just wanted to say that I am very happy to read this thread.

I have written several disparaging posts about AdaCore and my take on the situation was:

- GNAT Pro: professional support, more targets.
- GNAT Community: Lower level of quality assurance, less targets, and a pure GPL license for the run-time for a demoware experience
- GNAT FSF: least quality assurance designed to push users to the community build or Pro ASAP

I hope I was wrong all along and either way, the future just got a little brighter as AdaCore's two offerings will now be suitable for free or non-free software.

I don't believe that there is anything illegal about re-distributing GNAT Pro so if the gap in quality was so large between Pro and FSF, I think one paying customer might take pity on us eventually and release Pro to the world and reset the gap for a while.

With dark days setting in for the avionics industry (for a while at least), maybe AdaCore will eventually reconsider and will release Pro as their FSF offering to broaden their user base.

From: Simon Wright <simon@pushface.org>
Date: Wed, 29 Jul 2020 17:44:30 +0100

> I have written several disparaging posts about AdaCore and my take on the situation was: [...]

IMO you've been wrong all along, at least so far as the quality is concerned. Assurance, yes; number of targets, yes; support, yes.

> [...] I think one paying customer might take pity on us eventually and release Pro to the world and reset the gap for a while.

I wouldn't have done this; but we were stuck on an old release for a long time, so wouldn't have helped.

> With dark days setting in for the avionics industry (for a while at least), maybe AdaCore will eventually reconsider and will release Pro as their FSF offering to broaden their user base.

The only difference between the pro compiler and FSF is a few months.

From: Kevin K <kevink4@gmail.com>
Date: Sat, 15 Aug 2020 09:38:23 -0700

I hadn't seen this before today. The main impediment to me with the free version compared to the community edition is that with the community edition AdaCore took a set of packages that build together correctly. I tried to build a set of components from the free version (gcc 8.2 and later), gprbuild, etc. At the time, the other important components didn't all build successfully. Some components were ahead of others. So I wasn't able to build, for example, gps.

From: Roger Mc <rogermcm2@gmail.com>
Date: Sat, 15 Aug 2020 19:24:06 -0700

> I hadn't seen this before today. The main impediment to me with the free version compared to the community edition is that with the community edition AdaCore took a set of packages that build together correctly. [...]

Unfortunately, the AdaCore community 2020 edition doesn't include gps so I am currently using the community 2020 tool-chain and gps from community 2019. I did try to build gps from the current AdaCore community source but was unsuccessful. The main problem being that AdaCore seems to be in the midst of doing the necessary upgrade from Python 2 to Python3. I did attempt to do Python3 modifications myself but eventually got to a stage where I could proceed no further.

From: "Luke A. Guest" <laguest@archeia.com>
Date: Sun, 16 Aug 2020 03:42:11 +0100

> I hadn't seen this before today. The main impediment to me with the free version compared to the community edition is that with the community edition AdaCore took a set of packages that build together correctly. [...]

You'll have that issue on FSF because AdaCore don't tag for FSF releases like they should.

Having CE available is a massive mistake, one which they are realising far too late, imo.

From: Simon Wright <simon@pushface.org>
Date: Sun, 16 Aug 2020 11:08:18 +0100

> Unfortunately, the AdaCore community 2020 edition doesn't include gps

I _think_ this is on macOS? i.e. Linux, Windows include it? (presumably under its new name GNATstudio (modulo capitalisation))

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sun, 16 Aug 2020 05:08:13 -0700

> > Unfortunately, the AdaCore community 2020 edition doesn't include gps

> I _think_ this is on macOS? i.e. Linux, Windows include it? (presumably under its new name GNATstudio (modulo capitalisation))

Windows has <gnat>/bin/gnatstudio.exe

From: Roger Mc <rogermcm2@gmail.com>
Date: Sun, 16 Aug 2020 05:54:29 -0700

> I _think_ this is on macOS? i.e. Linux, Windows include it? (presumably under its new name GNATstudio (modulo capitalisation))

Yes. My system is macOS.

The source for GPS doesn't appear available from the AdaCore community version for any platform?

The source that I tried to build from was obtained from GIT.

I can only find GNATstudio under Ada core pro. Is it available elsewhere?

From: Stéphane Rivière <stef@genesix.fr>
Date: Mon, 17 Aug 2020 10:51:34 +0200

> You'll have that issue on FSF because AdaCore don't tag for FSF releases like they should.

> Having CE available is a massive mistake, one which they are realising far too late, imo.

Fully agree.

For the records, GVD (Gnu Visual Debugger) was buildable (under Windows[1] or Linux) but I _never_ succeeded to build GPS...

[1] For the now deprecated AIDE https://stef.genesix.org/aide/aide.html

From: Simon Wright <simon@pushface.org>
Date: Wed, 19 Aug 2020 15:29:00 +0100

> I did try to build gps from the current AdaCore community source but was unsuccessful. The main problem being that AdaCore seem to be in the midst of doing the necessary upgrade from Python 2 to Python3. I did attempt to do Python3 modifications myself but eventually got to a stage where I could proceed no further

I've reached the same stage. I can manage some of the 2-to-3 fixes (not the one in gobject-introspection, though), but the real problem for me is that there isn't a consistent complete set of sources, and some aren't provided on the AdaCore community site (e.g. pygobject, langkit, libadalang, libadalang-tools, ada_language_server). And, so far as I can see, langkit (20.2) isn't consistent with libadalang (20.2). And, my Python venv has got screwed.

Netflix & Twitter.

From: Andreas Zeurcher <ZUERCHER_Andreas@outlook.com>
Date: Wed, 19 Aug 2020 11:09:35 -0700

> I've reached the same stage. [...]

If multiple well-skilled people cannot build a GPL-licensed source code with the source code as provided and instructions as provided, wouldn't that be a black-&-white flagrant violation of the GPL? The natural conclusion seems to be: either the source code provided mismatched or the narrative instructions to build were omitting some secret-sauce, either of which was an unintentional or intentional preventative of success. The unintentionality versus intentionality would be able to be determined only after the fact by observing the root-cause of the preventative of successful building once that root cause is discovered/reported. This irreproducibility is both notable and highly interesting.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Wed, 19 Aug 2020 21:11:31 +0200

> If multiple well-skilled people cannot build a GPL-licensed source code with the source code as provided and instructions as provided, wouldn't that be a black-&-white flagrant violation of the GPL?

What else you expect when GTK and Python are used? GTK is practically impossible to bootstrap. Python is full of bugs and incompatibilities. On top of that for some mysterious reason AdaCore decided to use config scripts. No wonder it is a nightmare anywhere outside Linux.

If you think that commercial code delivered in sources is any better, you are wrong. Building from sources working out of the box is a rare exception. Most vendors simply check out the code from the repository and send it to you. They have no resources or desire to supply you with a working toolchain tailored for your targets, nor have they necessary knowledge anyway.

From: "Luke A. Guest" <laguest@archeia.com>
Date: Wed, 19 Aug 2020 21:21:20 +0100

> [...] No wonder it is a nightmare anywhere outside Linux.

I build on Linux and see my previous comment.

From: Simon Wright <simon@pushface.org>
Date: Wed, 19 Aug 2020 21:17:43 +0100

> [...] This irreproducibility is both notable and highly interesting.

Well - I have every sympathy with people who make a binary release and then move on in staggered stages, aiming for another binary release in a year's time. The sort of problem you encounter is that the the version of gobject-introspection on the CE site won't compile with Python 3.8.5, because of the removal of the DL_EXPORT macro that was deprecated with Python 2.3; while libadalang _requires_ Python 3.8.5. The latest glib uses Yet Another Build Tool (meson/ninja), and the script doesn't export a header required by gtk-3.14+ ... it's not so much DLL Hell as a version compatibility tightrope.

From: "Luke A. Guest" <laguest@archeia.com>
Date: Wed, 19 Aug 2020 21:19:50 +0100

> If multiple well-skilled people cannot build a GPL-licensed source code with the source code as provided and instructions as provided [...]

Every component from AdaCore is an absolute fucker to build.

From: Simon Wright <simon@pushface.org>
Date: Wed, 19 Aug 2020 21:50:29 +0100

> Every component from AdaCore is an absolute fucker to build.

The "front-line" components (gnatcoll*, gprbuild, xmlada & friends) build pretty reliably for me, even taking the master branch.

From: "Luke A. Guest" <laguest@archeia.com>
Date: Wed, 19 Aug 2020 22:35:07 +0100

> The "front-line" components (gnatcoll*, gprbuild, xmlada & friends) build pretty reliably for me, even taking the master branch.

That's only after you work out which commit to build, after many attempts at building.

From: <steve@cunningsystems.com>
Date: Wed, 19 Aug 2020 23:12:07 -0700

> I think that I also found this. Sometimes, the master worked, occasionally the stable (?) worked but I'd sometimes (often?) have to resort to an earlier build to achieve success!

Note that the libadalang/master and langkit/master repositories are often out ahead of gps/master. You'll have better luck if you build them from stable branches, then gps/master should build.

Similarly spark2014/master is often ahead of FSF gcc/master. gcc/master is usually in sync with spark2014/fsf branch.

From: Roger Mc <rogermcm2@gmail.com>
Date: Wed, 19 Aug 2020 22:42:24 -0700

> [...] the real problem for me is that there isn't a consistent complete set of sources [...]

I think I managed to do all the Python 3 conversions but couldn't get linking to work.

I recall that getting the prerequisites built was a challenge and found that if I changed anything in any of them I'd have to go back and start building them from scratch again. Worse, I seem to recall that for at least one of them, probably langkit, I'd have to delete it and reload it from its archive file.

It's comforting to find that many of the opinions expressed in this thread are similar to my own.

[Around this point, the thread veers off towards Python specifics, although with a relation to Ada features. --arm]

From: Roger Mc <rogermcm2@gmail.com>
Date: Wed, 19 Aug 2020 22:48:34 -0700

> What else you expect when GTK and Python are used? [...]

Incredibly, Python seems to be the language of choice for "teaching" the now-defunct discipline of software development, even by leading universities as far as I can discover.

Most of the rules of disciplined software development seem to have been discarded long ago. In particular, the maintainability aspect seems to have disappeared.

From: Stéphane Rivière <stef@genesix.fr>
Date: Thu, 20 Aug 2020 08:43:13 +0200

> Incredibly, Python seems to be the language of choice for "teaching" the now-defunct discipline of software development, even by leading universities as far as I can discover.

It's not incredible. It's led to 737max failure, It's Idiocracy.

RM about Idiocracy could be the movie Idiocracy. I urge you to see it. Its nickname is: the movie which has become a documentary. It's delightfully vulgar but above all incredibly relevant and funny.

https://en.wikipedia.org/wiki/Idiocracy

In any case, AdaCore's iechoice of python is miserable. It would have been wiser, if there was a need for a scripting language, to implement a subset of Ada (like Gautier de Montmollin HAL :) For the doc, they even abandoned GNU/Texinfo for Python...

Moreover, the GPS code has always been problematic. I remember the first versions where a very large portion of the code was in C because they had integrated a full version of an old version of berkeley DB...

From: Vincent Diemunsch <vincent.diemunsch@gmail.com>
Date: Mon, 31 Aug 2020 07:54:43 -0700

> In any case, AdaCore's choice of python is miserable. It would have been

> wiser, if there was a need for a scripting language, to implement a

> subset of Ada (like Gautier de Montmollin HAL :) For the doc, they even

> abandoned GNU/Texinfo for Python...

I agree.

And they also used Python for libadalang: "libadalang is using the Langkit framework as a basis, and is at the time of writing the main project developed using it. The language specification, while embedded in Python syntax, is mostly its own language, the Langkit DSL, that is used to specify the part of Ada syntax and semantics that are of interest to us."

I wonder if it would have been possible to create a library of objects directly in Ada, somehow equivalent in features to Python's Objects, but with the advantage of strong typing and a compilation to native instructions. It would require a major use of interfaces, one for each Python's built-in type class, but it would have been a foundation for many other applications.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Tue, 1 Sep 2020 11:22:04 -0700

> And they also used Python for libadalang [...]

> I wonder if it would have been possible to create a library of objects directly in Ada, somehow equivalent in features to Python's Objects, but with the advantage of strong typing [...]

Langkit uses the advanced features of Python to create a Domain Specific Language (DSL) for defining Abstract Syntax Trees. The DSL also defines much of the user API for accessing the syntax tree after parsing. You could accomplish something similar by using a grammar generator (WisiToken or Langkit :) to create a parser for the desired DSL (as WisiToken does), but then defining the API would be done separately, and the correspondence between the API and the syntax tree maintained manually (ie error-prone). I think Python is a good choice for this application - there are probably other languages with similar features that could have been used, but Ada is not one.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Tue, 1 Sep 2020 20:59:52 +0200

> Langkit uses the advanced features of Python to create a Domain Specific Language (DSL) for defining Abstract Syntax Trees.

Which is a big mistake, as well. Each intermediate is yet another point of error.

> I think Python is a good choice for this application - there are probably other languages with similar features that could have been used, but Ada is not one.

I doubt there could exist applications for languages like Python. Anyway, GPS is demonstratively not.

I also do not believe in heavily scripted IDEs. I certainly do not want GPS becoming Emacs. Any usability GPS has, comes from not being Emacs, or, for that matter, Visual Studio with its horrific VB scripts.

Proposal: Auto-allocation of Indefinite Objects

From: Yannick Moy <moy@adacore.com>
Subject: Re: Proposal: Auto-allocation of Indefinite Objects
Date: Mon, 27 Jul 2020 00:47:30 -0700
Newsgroups: comp.lang.ada

[This thread continues from AUJ 41.2, on the topic of having a mechanism to transparently allocate indefinite objects as if they were definite, e.g., as record members. --arm]

Hi Stephen,

> My proposal is that it should (sometimes?) be possible to declare objects of indefinite types such as String and have the compiler automatically declare the space for them without the programmer having to resort to access types.

I agree with the goal.

> Benefits:

>

> 1. Easier, especially for newbies/students.

> 2. Safer due to reduced use of access types.

> 3. Remove the need to have definite and indefinite versions of generic units.

I agree with 2 only if we can combine this with safe handling of aliasing. It would be terrible to have such a feature lead to unsafe code if you somehow copy the pointer. Also, for strings that's possibly not the only change needed. What you'd like really is to be able to reassign the string to some larger/smaller string, like you do when using Unbounded_String.

On 2020-04-04, Jeffrey R. Carter wrote:

> the ARG is aware of them and has chosen to take no action. That seems unlikely to change.

On the other hand, AdaCore has launched a project to collect/discuss ideas/suggestions/problems regarding the evolution of Ada and SPARK: https://github.com/AdaCore/ada-spark-rfcs

Feel free to open an Issue there on that topic.

From: "J-P. Rosen" <rosen@adalog.fr>
Date: Mon, 27 Jul 2020 11:21:16 +0200

> I agree with the goal.

You have it already. It's called Unbounded_String.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Mon, 27 Jul 2020 11:49:28 +0200

> You have it already. It's called Unbounded_String.

Not really.

1. Unbounded_String is a compromise needed when the string length changes during its life. The great majority of cases allocate [and initialize] a string just once. [addressed to be the cases when using a discriminant does not work]
2. There is nothing for arrays that are not strings and for other indefinite types. E.g.:

```
type Node_Type is record
   Item : new Element_Type'Class;
   Prev : Node_Ptr_Type;
   Next : Node_Ptr_Type;
end record;
```

3. There is nothing for serialization and marshaling objects logically containing strings and other indefinite types.

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Mon, 27 Jul 2020 17:48:40 -0000

> My proposal is that it should (sometimes?) be possible to declare objects of indefinite types such as String and have the compiler automatically declare the space for them [...]

In one sense we already have this ... in that we can do this in a Declare block, where stack allocation is a practical implementation.

But what about cases where (for whatever reason) we want it allocated on the heap?

In another sense we have it as JP Rosen said, for the specific example Unbounded_String.

Is there any way we could generalise the (storage, access and lifetime aspects of) Unbounded_String for unconstrained arrays and discriminated records in such a way that Unbounded_String can be a simple instantiation of one of these?

But without the full flexibility (or overhead) of controlled types. So, somewhere in between, as:

1. Controlled type
+ 2. Unconstrained Array or
  + Discriminated Record
  + 3. Unbounded String (instance of 2)

2) can be implemented internally using pointers, but externally appears to be a data object, just like Unbounded_String does, with similar semantics.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Mon, 27 Jul 2020 22:02:57 +0200

> 2) can be implemented internally using pointers, but externally appears to be a data object, just like Unbounded_String does, with similar semantics.

No, the point is that Unbounded_String is exactly opposite to what is required. In no case it should appear as an object of a different type!

Compare access to string P with unbounded string U:

```
for I in P'Range loop -- This is OK
P(J) := 'a' -- This is OK
```

Now would you do:

```
To_String (U) (J) := 'a' -- Garbage!
```

What if the original object must be a class-wide object, task, protected object, limited object etc?

Ada's access types delegate all operations to the target object, except assignment. This is the key property that the proposal in my view must retain.

From: "Jeffrey R. Carter" <spam.jrcarter.not@spam.not.acm.org>
Date: Mon, 27 Jul 2020 22:31:34 +0200

> Is there any way we could generalise the (storage, access and lifetime aspects of) Unbounded_String [...]

Ada.Strings.Unbounded can be considered a combination of Ada.Containers.Indefinite_Holders instantiated for String and Ada.Containers.Vectors instantiated with Positive and Character, with some additional operations added.

The To_String and To_Unbounded_String operations of Unbounded_String are similar to the Element and Replace_Element operations of Holder, which do not exist for Vector.

The indexed operations of Unbounded_String are similar to the indexed operations of Vector, which do not exist for Holder.

If Ada.Containers.Vectors had an additional generic formal type

```
type Fixed is array (Index_Type range
  <>) of Element_Type;
```

and 2 new operations

```
function To_Fixed (From : Vector)
  return Fixed;
function To_Vector (From : Fixed)
  return Vector;
```

then we wouldn't need Ada.Strings.Unbounded.

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Tue, 28 Jul 2020 14:28:53 -0000

> No, the point is that Unbounded_String is exactly opposite to what is required. In no case it should appear as an object of a different type! [...]

> To_String (U) (J) := 'a' -- Garbage!

That wasn't the aspect of Unbounded I was getting at. I agree ... garbage.

What I meant was that Unbounded doesn't load New, dereferencing, deallocation etc onto the programmer, but hides the access details, and our indefinite type should do the same (the compiler can probably to a better job than the programmer anyway).

I'm suggesting something more like the C++ reference, signalling (perhaps by adding a reserved word "indefinite") that fixed size allocation won't work; and implementation is more in line with a controlled type but with system-provided Initialise,Adjust,Finalize providing the required operations (no need for the programmer to provide them).

```
A : String := "hello" -- a definite string
P : access String := new String'("hello");
Q : indefinite String := "hello";
...
  begin
    for I in P'Range loop -- This is OK
      P(J) := 'a'; -- This is OK
      Q(J) := 'a'; -- also OK. But index out of
            --range would raiseConstraint Error

...

    Q := "hello_world"; -- deallocates,
    -- allocates with new bounds

...

  end;   -- deallocate Q here.
```

It follows that "indefinite" cannot also be "aliased" unless we want to implement smart pointers. For simplicity I'd suggest disallowing "aliased indefinite" on the grounds that "access" can (should) be used instead.

Records (including tagged, class wide, discriminated) should work the same, but probably with shallow copy on assignment if they contain access types.

If there is no re-allocation (no different size assignment) the compiler is free to substitute direct (stack) storage instead of heap allocation and implicit access types. So for example instead of

```
    A : constant String := "done";
 ...
  loop
    declare
      P : String := Get_Line;
    begin
         exit when P = A;
    end;
  end loop;

  A : constant String := "done";
  Q : indefinite String;
 ...
  loop
    Q := Get_Line;
    exit when Q = A;
  end loop;
```

the implementation can be either an implicit declare block or an implicit access type. However, where Q has several reassignments within a block, and the compiler can't determine the size, an implicit access type must be used. (If it can, it can warn that "indefinite " is unnecessary).

> What if the original object must be a class-wide object, task, protected object, limited object etc?

> Ada's access types delegate all operations to the target object, except assignment. This is the key property that the proposal in my view must retain.

Indefinite can also be applied to records (discriminated, class wide, etc) here the size is indeterminate and may vary on reassignment. Assignment would always be shallow copy (where the record contained access types).

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Tue, 28 Jul 2020 16:59:09 +0200

> I'm suggesting something more like the C++ reference, signalling (perhaps by adding a reserved word "indefinite") that fixed size allocation won't work;

Equivalent of C++ reference in Ada is renaming.

> Q : indefinite String := "hello";

I think the keyword is misleading. Maybe this:

```
  Q : new String := "hello";
```

And I don't like initialization. It was a mistake to have limited return. The syntax must stress that all initialization is strictly in-place. No copies involved because the pool is fixed.> ...

> begin

> Q := "hello_world"; -- deallocates, allocates with new bounds

> ...

> end; -- deallocate Q here.

The rule could be "same pool" as of the container. In the case of a block, the pool is the stack. In the case of a record member, the pool is the pool of where the record itself is allocated. So that you could allocate all [the full] object in the same pool.

> It follows that "indefinite" cannot also be "aliased" unless we want to

> implement smart pointers. For simplicity I'd suggest disallowing "aliased

> indefinite" on the grounds that "access" can (should) be used instead.

It makes sense, but there are use cases for having it aliased:

```
  X : indefinite T;
  Y : indefinite S (X'Access);
      -- Access discriminant
```

[...]

> Assignment would always be shallow copy (where the record contained access types).

That would be inconsistent. IMO, it should be a deep copy, provided such a component would not make the type limited, of which I am not sure.

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Wed, 29 Jul 2020 15:33:33 -0000

> Equivalent of C++ reference in Ada is renaming.

OK. Not quite sure how complete the correspondence between reference and renaming is, but I can see similarities.

[...]

> The rule could be "same pool" as of the container. In the case of a block, the pool is the stack. In the case of a record member, the pool is the pool of where the record itself is allocated. So that you could allocate all object in the same pool.

Looks like a good rule. Saves the compiler having to plant deallocations if the whole pool is to be de-allocated.

[...]

>> Assignment would always be shallow copy (where the record contained access types).

> That would be inconsistent. IMO, it should be a deep copy, provided such a component would not make the type limited, of which I am not sure.

Honest question: Inconsistent with what? I suggested shallow copy just for simplicity, and for no (ahh) deeper reason.

But again, I'm probably missing something.

Thank you for your thoughts. I don't know if this is worth developing into an AI.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Wed, 29 Jul 2020 18:20:24 +0200

[...]

In general, there are two close but not equivalent objectives: one is handling indefinite components of records; another is a transparent holder object integrated into the language (without generic mess).

Your use case is about the latter. My is rather the former.

I doubt it is possible to unite both objectives in a single AI.

On 29/07/2020 17:33, Brian Drummond wrote:

> I suggested shallow copy just for simplicity, and for no (ahh) deeper reason. But again, I'm probably missing something.

If you make a shallow copy of

```
type Node_Type is record
   Item : new Element_Type;
   Prev : Node_Ptr_Type;
   Next : Node_Ptr_Type;
end record;
```

you create a dangling pointer should the original node disappear. A deep copy would create a new target for new Item.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Thu, 30 Jul 2020 20:28:32 +0200

[...]

> The compiler should be able to determine if [...] the use of Q (the indefinite type) is equivalent to a Declare block (i.e. can be on the stack; new stack frame in each iteration; no relocation ever required) or not.

I don't want the compiler deciding where Q is allocated, especially because this could break things:

1. Large object moved to the stack

2. Lock-free code starting using heap lock when moved from the stack.

The mechanism should be transparent. I do not like Unbounded_String for many reasons. Fiddling with the heap is one of them. I do not know which heuristic it uses to reduce reallocation and how much extra memory it takes under which circumstances.

[...]

From: "Randy Brukardt" <randy@rrsoftware.com>
Date: Sun, 9 Aug 2020 19:31:06 -0500

> I don't like compiler relocating objects. If the pool is a stack (or heap organized as a stack) it might be unable to do this.

This is not that hard to deal with. Janus/Ada handles discriminant-dependent components of mutable objects this way: they are allocated on the stack, but if they have to be reallocated they move to the heap.

I note that the original idea already exists for discriminant-dependent components -- that's a bit more painful to use but hardly difficult. The main issue is that most compilers fail to support these components properly, using some sort of max-size implementation unconditionally rather than switching to a pool-based implementation when the max size is too large. I've never understood why Ada compilers were allowed to make such a limitation (it becomes a major limitation when working on non-embedded programs), while similar limitations on case statements and aggregates are not allowed.

From: "Randy Brukardt" <randy@rrsoftware.com>
Date: Sun, 9 Aug 2020 19:39:31 -0500

> I don't want the compiler deciding where Q is allocated, especially because this could break things:

> 1. Large object moved to the stack

The compiler is buggy IMHO if this breaks something. Any compiler has to be able to deal with objects that exceed the maximum stack frame, and move those to somewhere that they will fit (or reject completely).

Yes, most compilers are buggy this way (including mine in a few cases). So what?

> 2. Lock-free code starting using heap lock when moved from the stack.

Expecting a compiler not to use the heap is silly in any case (outside of the No_Heap restriction - use that in Janus/Ada and the compiler refuses to do anything outside of elementary types). The compiler is supposed to be making the programmer's life easier, not adding new hurdles.

> I do not know which heuristic it uses to reduce reallocation and how much extra memory it takes under which circumstances.

That's the idea of such mechanisms. If you really need control, you do not use these abstractions and instead write the stuff yourself explicitly using access types and the like.

Otherwise, you use containers and unbounded strings, and they do what they do. There's no free lunch. But the need to be explicit should be very rare - the main problem is programmers with insufficient trust that a compiler will do the right thing.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Mon, 10 Aug 2020 10:57:44 +0200

> That's the idea of such mechanisms. If you really need control, you do not use these abstractions and instead write the stuff yourself explicitly using access types and the like.

Right, that is my take on the proposal. If I am ready to compromise on #1 and #2, I can use an abstraction hiding pool access. Otherwise I want a language construct being more safe than raw access types.

> Otherwise, you use containers and unbounded strings, and they do what they do.

No, from the abstraction point of view they do not. They indeed abstract the memory allocation aspect, but they do that at the cost of *everything* else. Unbounded_String is no string anymore. Container is neither array nor record type. Unbounded_String must be converted forth and back. For containers I must use ugly hacks like iterators to make them resemble arrays and records introducing whole levels of complexity to fight through every time the compiler or I miss something.

In most cases I prefer to keep a clear array or record interface at the expense of manual memory management.

> There's no free lunch.

I think with a better type system there could be a whole banquet. (:-))

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Mon, 10 Aug 2020 10:58:20 +0200

> Janus/Ada handles discriminant-dependent components of mutable objects this way: they are allocated on the stack, but if they have to be reallocated they move to the heap.

What do you do if such an object is allocated via pool-specific access type?

[...]

From: "Randy Brukardt" <randy@rrsoftware.com>
Date: Wed, 19 Aug 2020 19:10:27 -0500

> I think with a better type system there could be a whole banquet. (:-))

Maybe. but IMHO a better type system would get rid of arrays and strings altogether and only have containers/records of various sorts. The complexity of having both solving the same problems (not very well in the case of arrays/strings) doesn't buy much. I suspect that a user-defined "." as you've proposed elsewhere would eliminate most of the rest of the problems (and unify everything even further).

*From: "Randy Brukardt"
<randy@rrsoftware.com>
Date: Wed, 19 Aug 2020 19:13:15 -0500*

> What you do if such an object is allocated via pool-specific access type?

The whole object goes in that pool. The entire mechanism in Janus/Ada is built around pools - the stack is represented by a pool object as well as various other pools to support the mechanism.

[...]

*From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Date: Thu, 20 Aug 2020 19:49:34 +0200*

> The whole object goes in that pool. [...]

OK, but then you are back to the problem that you do not know how that pool works. The user pool might require a certain order of objects inside it and your interference with relocation will break it.

[...]

*From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Date: Thu, 20 Aug 2020 19:49:44 +0200*

> [...] IMHO a better type system would get rid of arrays and strings altogether and only have containers/records [...]

But records and arrays are needed as building blocks of containers. How would you get rid of them?

*From: Dennis Lee Bieber
<wlfraed@ix.netcom.com>
Date: Thu, 20 Aug 2020 16:19:52 -0400*

>But records and arrays are needed as building blocks of containers.

And likely needed for any embedded or low-level work where they are mapped to things like (GP) I/O ports or such...

*From: "Randy Brukardt"
<randy@rrsoftware.com>
Date: Thu, 20 Aug 2020 18:25:46 -0500*

> [...] The user pool might require a certain order of objects inside it and your interference with relocation will break it.

Such a pool does not implement the interface as defined in 13.11. It's OK of course to write a pool that depends on implementation-specific properties (I've done it many times), but such a pool is not usable with portable Ada code. If the pool allows any sort of allocation at any time, then it will work just fine with the Janus/Ada implementation.

[...]

Note that this is the reason that Ada doesn't support specifying the pool used by a container. It would not be reasonable to restrict the allocations in any way, so implementation-dependent pool designs would not work.

[...]

*From: "Randy Brukardt"
<randy@rrsoftware.com>
Date: Thu, 20 Aug 2020 18:30:07 -0500*

> But records and arrays are needed as building blocks of containers. How would you get rid of them?

There's no reason that a compiler couldn't "build-in" a simple bounded vector container as the basic building block. We already do that for things like Ada.Exceptions, Unchecked_Conversion, and Unchecked_Deallocation, so it's no harder to do that for a vector. (Probably would need some sort of fixed vector for interfacing purposes as well, to deal with other language's and/or system's memory layout.)

One could do something similar for records, although I would probably leave them as in Ada and just allow user-definition of "." (via a getter/setter pair).

*From: "Randy Brukardt"
<randy@rrsoftware.com>
Date: Thu, 20 Aug 2020 18:33:40 -0500*

> And likely needed for any embedded or low-level work where they are mapped to things like (GP) I/O ports or such...

Yes, a fixed vector container would be needed for interfacing (probably wouldn't use it for anything else). But there's no reason that can't be provided as a container, so long as representation guarantees (esp. Component_Size) are included. Remember that containers (in Ada 202x) have indexing, aggregates, and all of the useful basic operations. The stuff that's missing is the same stuff that adds a vast amount of complexity to Ada (and possibilities for bugs) - hardly anyone would miss it.

*From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Date: Fri, 21 Aug 2020 08:46:07 +0200*

> There's no reason that a compiler couldn't "build-in" a simple bounded vector container as the basic building block.

That simply replaces the word "array" with four words "simple bounded vector container." The construct is still there and it is still built-in. The syntax and usability are drastically worse, though.

> One could do something similar for records, although I would probably leave them as in Ada and just allow user-definition of "." (via a getter/setter pair).

*From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Date: Fri, 21 Aug 2020 09:08:46 +0200>*

>> I meant that if you used a pool behind the scenes for local objects you could do that task-specific eliminating interlocking.

> Whether that would be worthwhile would depend on how expensive the locking is.

It could be very expensive on a multi-core architecture. I also think about scenarios when the object is used inside a protected action. I would not like to see any pool interaction in an interrupt handler!

*From: "Randy Brukardt"
<randy@rrsoftware.com>
Date: Sat, 22 Aug 2020 23:48:13 -0500*

> That simply replaces the word "array" with four words "simple bounded vector container." The construct is still there and it is still built-in. The syntax and usability are drastically worse, though.

??? The syntax of use is the same (as it is in Ada 2012). Declaration would be an instance, about the same length and wordiness as an array declaration. Yes, junk like slices, settable/retrievable bounds, and built-in operations that are rarely used would be gone, but so would the rather substantial overhead that those things entail. There'd be a lot more flexibility in implementation, which would allow better implementations.

Virtually every array that I write has a fixed size (capacity really) and a usage high-water mark (a "length"). Having that generated automatically would be usually better than having to reinvent it literally every time I program something. (And as you've noticed repeatedly, Ada's type abstraction isn't good enough to make it practical to build anything reusable to do that.)

>> I would probably leave them as in Ada and just allow user-definition of "."

???

The basic idea would be to eliminate the huge number of special cases that exist in Ada resolution and essentially make *everything* a subprogram call at its heart. Ada did that for enumeration literals and that model makes sense for pretty much everything: object usage, indexing, selection, etc. It would be much easier to prove that resolution is doing the right thing (I don't think that would be practically possible for Ada).

*From: "Randy Brukardt"
<randy@rrsoftware.com>
Date: Sat, 22 Aug 2020 23:52:30 -0500*

> Really? I would miss array conversions, slices, equivalence of same length index ranges, constrained array subtypes etc.

Those things are mostly useful for making work for programmers. Note that I'm assuming that Strings are a completely separate abstraction - a UTF-8 string is not an array and shouldn't be treated as one. (Indexing of individual characters being very expensive.) Fixed constrained arrays would be available for interfacing (they're not really useful for much else). Note that a bounded vector is allocated statically, so there's no extra cost to using it (unlike an unbounded vector or string).

From: "Randy Brukardt" <randy@rrsoftware.com>
Date: Sun, 23 Aug 2020 00:03:01 -0500

> [...] I also think about scenarios when the object is used inside a protected action. I would not like to see any pool interaction in an interrupt handler!

Interrupt handlers shouldn't be doing anything other than unblocking tasks. I think it is a mistake to allow anything else (as there are always problems with race conditions if you do so). So no heap possibilities as very little is going on.

Available Ada Compilers

From: gdotone@gmail.com
Subject: Is there another ada compiler
Date: Sun, 2 Aug 2020 19:22:10 -0700
Newsgroups: comp.lang.ada

Is there another Ada compiler other than AdaCore?

From: gautier_niouzes@hotmail.com
Date: Mon, 3 Aug 2020 00:54:26 -0700

Check here: http://unzip-ada.sf.net/ #adacomp

or here https://www.adaic.org/ ada-resources/pro-tools-services/

for instance.

If you are looking for another *open-source* compiler, but rather incomplete:

https://hacadacompiler.sourceforge.io/

From: Shark8 <onewingedshark@gmail.com>
Date: Mon, 3 Aug 2020 06:51:54 -0700

RR Software has Janus/Ada.

PTC has ObjectAda and ApexAda.

Green Hills has an Ada compiler.

DDC-I has a compiler.

IBM used to have a compiler. (I'm not sure they do any more.)

There's also work being done on some open source compilers like HAC or my own Byron.

From: "Luke A. Guest" <laguest@archeia.com>
Date: Mon, 3 Aug 2020 15:29:34 +0100

> RR Software has Janus/Ada.

> PTC has ObjectAda and ApexAda.

> Green Hills has an Ada compiler.

> DDC-I has a compiler.

> IBM used to have a compiler. (I'm not sure they do any more.)

They sold it.

All the above are commercial and cost £££££'s

> There's also work being done on some open source compilers like HAC or my own Byron.

HAC is not going to be a full compiler, so it's really worth mentioning.

Byron's not anywhere near close to generating assembly.

In answer to the OP's question, no, there isn't another open source compiler.

From: nobody in particular <nobody@devnull.org>
Date: Mon, 3 Aug 2020 15:18:36 +0000

> RR Software has Janus/Ada.

Honest company run by good guy Randy Brukhard, who is a long time participant on the newsgroup. Unfortunately, not available on the platform I wanted it for. Hobbyist-friendly.

> PTC has ObjectAda and ApexAda.

There was a recent announcement here in the newsgroup, unfortunately without any pricing. Pricing is also not found on the PTC website. In the past, Aonix did have a hobbyist compiler but I haven't seen it for years.

> Green Hills has an Ada compiler.

Huge money and the salesman I spoke with displayed significant disdain when I turned out to be an individual rather than a company. Did not disclose pricing. However, in speaking with another participant off-list, I was given some sense of the pricing.

> DDC-I has a compiler.

Not sure if anything past '83 is supported. But do check if you're interested. I believe JOVIAL is also available from DDC-I.

> IBM used to have a compiler. (I'm not sure they do any more.)

It was sold to a company in Washington, D.C. which I believe still sells the Ada 95 compiler. I don't believe they support any additional standards after 95. I'm sorry, I can't remember the name.

I attempted to get a hobbyist distribution to run on the Hercules z/Architecture emulator (which also supports MVS, MVS/ESA, and OS/390) but was not successful. Appeared to be a reasonable guy and the product was well integrated in MVS/ESA but probably not generally useful to most people in this newsgroup. If it is, would be worth identifying the company and starting a dialog.

Lastly, we should mention gcc-ada which was still out there for Linux and Solaris last I looked, and even for some unusual platforms like Solaris SPARC. The SPARC platform maintainer was very helpful and I got a copy at some point, I can't remember but I think around gcc5.

There used to be GNAT 3.15p (last non-GPL) release but it was cruelly excised from all servers and download sites when AdaCore happened.

We should note, GNAT / AdaCore were created on the backs of American taxpayers via a grant to New York University. Unfortunately, the taxpayers got the shaft and a profitable business was born to continue the fun.

From: Micronian Coder <micronian2@gmail.com>
Date: Mon, 3 Aug 2020 10:37:05 -0700

Just because a compiler is not free does not mean it is not relevant to someone. In addition, the OP did _not_ specifically ask for an open source compiler. They asked if there are other compilers. Hobbyists generally want a free compiler, so by default GNAT is the one that is used. For companies who want commercial support and are fine with paying money, then the other options are perfectly fine.

Of the commercial ones listed, Janus/Ada is the more affordable one for an individual willing to spend money (see http://www.rrsoftware.com/html/companyinf/prices.htm), especially if they are a student (http://www.rrsoftware.com/html/companyinf/educ.htm). While it's not as up to date as GNAT in terms of Ada2012 support, it's still enough to develop software with (note: Windows only which is fine for many people and can probably run on Wine for Linux users). Judging by Randy's posts in this group, one can expect good support from RRSoftware.

I should point out that PTC is known to provide free access to their compilers if it is for developing *open source* Ada software. Gautier has confirmed this on Reddit (https://www.reddit.com/r/ada/comments/hw33kr/ptc_objectada_for_windows_version_102_outprovides/fyzgpss?utm_source=share&utm_medium=web2x). So there is potential

From: gautier_niouzes@hotmail.com
Date: Mon, 3 Aug 2020 14:44:43 -0700

> We should note, GNAT / AdaCore were created on the backs of American taxpayers via a grant to New York University. [...]

It's called public-private partnership ;-)

See Tesla or SpaceX for other examples. Is it so bad?

BTW, weren't most early Ada vendors essentially financed by the US DoD?

As a consolation, consider that the American taxpayers have become (at least for a while) minority contributors to the US budget...

From: Andreas Zuercher <ZUERCHER_Andreas@outlook.com>
Date: Mon, 3 Aug 2020 17:23:59 -0700

> We should note, GNAT / AdaCore were created on the backs of American taxpayers via a grant to New York University. [...]

Well, I am not usually in the habit of saying nice things about GNAT, but let us compare FSF's GCC GNAT with FSF's GCC CHILL. Ada and CHILL are fierce competitor languages: one from NATO military and the other from ITU-T telecom, where Ada trended a little more toward Wirth family of languages as inspiration whereas CHILL trended a little more toward PL/I as inspiration. Both languages had a 2-decade mandate to be utilized in their respective industrial sectors, but each's mandate had evaporated by the latter half of the 1990s.

Ada had AdaCore arise through several mergers as the for-profit support company for open-source software, analogous to Cygnus Solutions during the 1990s, and its acquirer RedHat until this day. CHILL had a different business model entirely. CHILL compilers were produced by the telecom companies that were self-mandated to use CHILL. If Ada had that business model, Raytheon would have authored its own compiler, Lockheed-Martin would have authored its own compiler, Boeing would have authored its own compiler, Airbus would have authored its own compiler, and so forth. Eventually the telecom companies in Europe fatigued of the effort needed to write a compiler for an evolving language standard (ITU-T Z.200 and ISO 9496), so 2 of them (Alcatel or Siemens, IIRC) outsourced their internal compiler development to Per Bothner, who eventually landed at Cygnus Solutions, after University of Wisconsin at Madison (years after Randy). Eventually, Cygnus Solutions convinced FSF to allow their CHILL compiler into GCC.

Shortly after FSF GCC admitted CHILL into its compiler suite, RedHat bought Cygnus Solutions and nearly all of the European telecom companies were finalizing the financially painful governmental reform where PTTs (postal-telephone-telegraph agencies of governments) were divesting their relationship with the equipment manufacturers—much like AT&T divested WesternElectric/Lucent and Bell Canada no longer had Northern Telecom as favorite-son supplier during much the same 1990s time period. Long story short, when FSF pleaded for someone anyone to update GCC CHILL to GCC 3.X internals, no one stepped forward to fund the effort with money, and most especially no one donated source code as in-kind support. GCC CHILL as donorware ended as of GCC 2.95.

Whatever or however one might critique FSF GNAT versus AdaCore GNAT Pro differences or delays or never achieving perfect congruence among any pairwise matching of any of their releases, GNAT's viability to continue maintenance & evolution is far better that CHILL's donorware-based approach that failed miserably under the same FSF GCC umbrella during the same time period. So matters could be far far worse than they are.

PolyORB and the DSA Annex

From: tonyg <tonythegair@gmail.com>
Subject: Polyorb abd the DSA Annex
Date: Mon, 3 Aug 2020 04:04:27 -0700
Newsgroups: comp.lang.ada

I just tried to build the git cloned copy of PolyORB (failed on the configure!) with the 2020 community version of GNAT. It said I had no GNAT Ada compiler. It was on the path and it pointed to the gcc compiler on the path. Are they compatible? Is there still a PolyORB "enthusiast" list ?

From: "Luke A. Guest" <laguest@archeia.com>
Date: Mon, 3 Aug 2020 12:40:32 +0100

Distributed annex is being removed from GNAT due to "lack of customer interest."

From: Shark8 <onewingedshark@gmail.com>
Date: Mon, 3 Aug 2020 06:47:12 -0700

This is pretty sad, and IMO, stupid; the ability to [relatively] easily make distributed applications via DSA is a killer feature and, in conjunction with Ada2020 'parallel' blocks/loops would make for a very attractive system.

IOW, the "lack of customer interest" is an excuse to shoot themselves in the foot.

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Mon, 3 Aug 2020 16:16:13 +0200

The reality is a bit more complex.

Distributed Annex is based on RPC.

Ada is largely used in the field applications, embedded, real-time. RPC are pretty much useless there, as well as in massively parallel applications.

For service-oriented sluggish applications RPC might be OK, but CORBA is a blocker there, because static topology/configuration is too rigid for such applications. (Static topology is less and less tolerated in the former as well)

P.S. I have an almost ready distributed Annex implementation based on inter process communication (no network, same box), but I have no information how to dock it into GNAT.

Dynamic Variable Creation a la PHP

From: Ian Douglas <ian@vionia.com>
Subject: Newbie question # 2
Date: Thu, 6 Aug 2020 11:40:35 -0700
Newsgroups: comp.lang.ada

I did try Google search and assorted books but could not find an answer.

In PHP, let's say we have a variable $fruit which contains the string "banana".

In PHP, if I do $$fruit, then it creates a variable $banana, which I can then do things with.

Does Ada support any such concept of taking the contents of one variable and using THAT as a variable?

I'm reading in a file which has the name of an object followed by some properties so I want to use the name as a variable ...

File is something I created, so it's not some random stuff, and the variables will be existing already.

From: Simon Wright <simon@pushface.org>
Date: Thu, 06 Aug 2020 19:56:23 +0100

I'd think of a record type to contain the properties, and then a map from object name to properties:

```
  type Properties is record
     Length : Positive;
     Width  : Positive;
  end record;
  package Object_Maps is new
Ada.Containers.Indefinite_Ordered_Maps
    (Key_Type     => String,
     Element_Type => Properties);
  Objects : Object_Maps.Map;
```

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 6 Aug 2020 22:20:11 +0300

> In PHP, let's say we have a variable $fruit which contains the string "banana". If I do $$fruit, then it creates a variable $banana, which I can then do things with.

Fortran has a similar feature, NAMELIST, for reading values into variables also named in the input. It can also be used for output.

Ada does not have such a feature.

[...]

From: Ian Douglas <ian@vionia.com>
Date: Thu, 6 Aug 2020 12:41:53 -0700

> I'd think of a record type to contain the properties, and then a map from object name to properties:

Yes, the variables are actually records.

>   package Object_Maps is new Ada.Containers.Indefinite_Ordered_Maps

Okay that's a new construct I haven't come across yet. Let me see what I can dig up on that.

From: Ian Douglas <ian@vionia.com>
Date: Thu, 6 Aug 2020 12:45:04 -0700

> Ada does not have such a feature.

I figured as much, probably "unsafe programming practice" at the end of the day.

[...]

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 6 Aug 2020 23:08:00 +0300

> I figured as much, probably "unsafe programming practice" at the end of the day.

I wouldn't say so. I think "namelist" input is a perfectly reasonable function to have in some programs, and is not particularly unsafe in any way -- if the programmer can limit the set of variables that can be named and changed by such input, which is the case in Fortran (and also in our various suggestions for implementing it in Ada).

PHP is (I believe) an interpreted language, so the symbol table is around at run-time, which makes it easy for PHP to support variables that refer to any other variable by its symbolic name. This can make "namelist" input in PHP unsafe, since the input can change any variable -- including variables that the programmer did not intend to be changeable in this way.

Ada is usually compiled, and the symbol table is not present when the compiled program runs, so it would be harder to implement a "namelist" input/output feature. But not impossible, as Fortran shows.

Ada on Beaglebone Black

From: Ricardo Brandão <rbrandao.br@gmail.com>
Subject: Running ADA on Beaglebone Black
Date: Sun, 9 Aug 2020 07:39:58 -0700
Newsgroups: comp.lang.ada

I've just acquired a Beaglebone black and I'm trying to run a simple program in Ada.

I tried to install GNAT but apt-get install GNAT or anything similar doesn't work.

I didn't find any place with the repository, nor any tutorial.

How is the best way to run Ada in the BBB?

From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>
Date: Sun, 9 Aug 2020 17:58:54 +0200

What distribution? APT suggests Debian or Ubuntu. In any case I would do a full upgrade to the latest version of either.

Under Debian buster you should do

```
apt install gnat-8
```

Under Ubuntu it would be

```
apt install gnat-10
```

And also:

```
apt install gprbuild
```

If these do not work check /etc/apt/sources.list

From: Philip Munts <philip.munts@gmail.com>
Date: Sun, 9 Aug 2020 10:17:12 -0700

Depending on what you want to accomplish (i.e. whether it needs the full Debian OS), and whether you are willing and able to do cross-compilation, I think that MuntsOS (https://github.com/pmunts/muntsos) is the easiest way to develop and run Ada programs on a BeagleBone or Raspberry Pi.

I have found that installing and maintaining and especially configuring I/O with Debian on a BeagleBone to be a pain in the nether regions. In contrast, I have attempted to make MuntsOS as easy to use as possible. Its only real downside is that it requires cross-toolchains running on a Linux (preferably Debian or Ubuntu) development host. Windows Subsystem for Linux (WSL1 or WSL2) works perfectly well for the development host, though.

From: Ricardo Brandão <rbrandao.br@gmail.com>
Date: Sun, 9 Aug 2020 12:56:35 -0700

> APT suggests Debian or Ubuntu. In any case I would do a full upgrade to the latest version of either.

Yes, I ran sudo apg-get update and GNAT appear in apt-cache search gnat

> Under Debian buster you should do

> apt install gnat-8

> And also:

> apt install gprbuild

I ran these commands and worked fine.

Thank you so much

MITRE's Top-25 List of 2020 Software-bug Categories

From: Andreas Zuercher <ZUERCHER_Andreas@outlook.com>
Subject: MITRE's top-25 list of 2020 software-bug categories
Date: Sat, 22 Aug 2020 09:31:13 -0700
Newsgroups: comp.lang.ada

https://www.bleepingcomputer.com/news/security/mitre-shares-this-years-top-25-most-dangerous-software-bugs/

Proper intended usage of Ada-specific features mitigates 9 of them, including a few that hit interpreted scripting languages hard. Others of the 25 are design-level almost independent of programming language. Still others of the 25 are cavalier/insufficient WWW-oriented string-processing or SQL string-processing or director-filename string-processing that could be conceivably done just as badly in Ada.

Conversely, if HOLWG were still pursuing their language-design goals today, certainly this MITRE* report would shape the design of an evolving GreenGreenerGreenest language today, instead of Ada solving primarily yesteryear's programming/software-engineering challenges well.

* defense contractor

From: Shark8 <onewingedshark@gmail.com>
Date: Tue, 25 Aug 2020 12:09:47 -0700

The interesting portion, in tabular form.

Rank - ID - Name - Score

1 CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') 46.82

2 CWE-787 Out-of-bounds Write 46.17

3 CWE-20 Improper Input Validation 33.47

4 CWE-125 Out-of-bounds Read 26.50

5 CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer 23.73

6 CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') 20.69

7 CWE-200 Exposure of Sensitive Information to an Unauthorized Actor 19.16

8 CWE-416 Use After Free 18.87

9 CWE-352 Cross-Site Request Forgery (CSRF) 17.29

10 CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') 16.44

11 CWE-190 Integer Overflow or Wraparound 15.81

12 CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') 13.67

13 CWE-476 NULL Pointer Dereference 8.35

14 CWE-287 Improper Authentication 8.17

15 CWE-434 Unrestricted Upload of File with Dangerous Type 7.38

16 CWE-732 Incorrect Permission Assignment for Critical Resource 6.95

17 CWE-94 Improper Control of Generation of Code ('Code Injection') 6.53

18 CWE-522 Insufficiently Protected Credentials 5.49

19 CWE-611 Improper Restriction of XML External Entity Reference 5.33

20 CWE-798 Use of Hard-coded Credentials 5.19

21 CWE-502 Deserialization of Untrusted Data 4.93

22 CWE-269 Improper Privilege Management 4.87

23 CWE-400 Uncontrolled Resource Consumption 4.14

24 CWE-306 Missing Authentication for Critical Function 3.85

25 CWE-862 Missing Authorization 3.77

From: Andreas Zuercher <ZUERCHER_Andreas@outlook.com>
Date: Tue, 25 Aug 2020 12:43:13 -0700

> Would 've been nice if you'd have also given the examples and how Ada solved them.

I am not going to write an entire textbook here on c.l.a, but here are the nine of the top twenty-five subcategories that I consider Ada diligently trying to mitigate or eliminate when properly utilized:

• 2nd-most frequent: CWE-787 Out-of-bounds Write

• 3rd-most frequent: CWE-20 Improper Input Validation

• 4th-most frequent: CWE-125 Out-of-bounds Read

• 5th-most frequent: CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer

• 8th-most frequent: CWE-416 Use After Free

• 11th-most frequent: CWE-190 Integer Overflow or Wraparound

• 13th-most frequent: CWE-476 NULL Pointer Dereference

• 17th-most frequent: CWE-94 Improper Control of Generation of Code ('Code Injection')

• 23rd-most frequent: CWE-400 Uncontrolled Resource Consumption

There are 1,248 Common Weakness Enumerations (CWEs) that MITRE lobs against software development (instead of against hardware development), so you can peruse the 26th through 1,248th if you so desire. Query 699 is the one for looking at the full inventory of subcategories of software defects. These 1,248 subcategories (and the aforementioned top-25 subcategories) fall into 40 more-macroscopic broader categories.

https://cwe.mitre.org/data/definitions/699.html

I claim that next-gen Ada (AdaNG, pronounced “a dang” as in do we give a dang or not) would use these 1,248 categories as measuring stick of expressibility of software-engineering correctness, just as HOLWG's Green and Ada used Steelman as measuring stick of the ability to express software-engineering correctness.

ptc® apexada | ptc® objectada®

Complete Ada Solutions for Complex Mission-Critical Systems

- Fast, efficient code generation
- Native or embedded systems deployment
- Support for leading real-time operating systems or bare systems
- Full Ada tasking or deterministic real-time execution

Learn more by visiting: **ptc.com/developer-tools**

ptc

Conference Calendar

Dirk Craeynest

KU Leuven, Belgium. Email: Dirk.Craeynest@cs.kuleuven.be

This is a list of European and large, worldwide events that may be of interest to the Ada community. Further information on items marked ♦ is available in the Forthcoming Events section of the Journal. Items in larger font denote events with specific Ada focus. Items marked with ☺ denote events with close relation to Ada.

The information in this section is extracted from the on-line *Conferences and events for the international Ada community* at http://www.cs.kuleuven.be/~dirk/ada-belgium/events/list.html on the Ada-Belgium Web site. These pages contain full announcements, calls for papers, calls for participation, programs, URLs, etc. and are updated regularly.

The COVID-19 pandemic had a catastrophic impact on conferences world-wide. Where available, the status of events is indicated with the following markers: "(v)" = event is held online, and "(h)" = event is held in a hybrid form (i.e. partially online).

2020

October 06-09 (v)	20th **International Conference on Runtime Verification** (RV'2020). Los Angeles, California, USA. Topics include: monitoring and analysis of the runtime behaviour of software and hardware systems. Application areas include cyber-physical systems, safety/mission critical systems, enterprise and systems software, cloud systems, autonomous and reactive control systems, health management and diagnosis systems, and system security and privacy.
October 19-24 (h)	18th **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2020). Hanoi, Vietnam. Topics include: theoretical and practical aspects of automated analysis, verification, and synthesis; program analysis and software verification; analytical techniques for safety, security, and dependability; testing and runtime analysis based on verification technology; analysis and verification of parallel and concurrent systems; verification in industrial practice; applications and case studies; automated tool support etc.
October 24-28 (v)	13th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2020). Porto, Portugal. ICST'2020 was postponed from 23-27 March to 24-28 October. Topics include: manual testing practices and techniques, security testing, model based testing, test automation, static analysis and symbolic execution, formal verification and model checking, software reliability, testability and design, testing and development processes, testing in specific domains (such as embedded, concurrent, distributed, ..., and real-time systems), testing/debugging tools, empirical studies, experience reports, etc.
October 26-27 (v)	14th **International Conference on Verification and Evaluation of Computer and Communication Systems** (VECoS'2020), Xi'an, China. VECoS'2020 was moved from 22-25 September in Xi'an, China, to 26-27 October in a virtual event format. Topics include: formal verification and evaluation approaches, methods and techniques, especially those developed for concurrent and distributed hardware/software systems, such as abstraction techniques, compositional verification, correct-by-construction design, rigorous system design, model-checking, performance and robustness evaluation, QoS evaluation, planning and deployment, dependability assessment techniques, RAMS (Reliability-Availability-Maintainability-Safety) assessment, verification & validation of IoT, verification & validation of safety-critical systems, worst-case execution time analysis, etc. Application areas include: communication protocols, cyber-physical systems, high-performance computing, internet of things, logistics systems, programming languages, real-time and embedded operating systems, telecommunication systems, etc.
November 02-06 (v)	IEEE **International Conference on Software Architecture** (ICSA'2020), Salvador, Brazil. ICSA'2020 was postponed from 16-20 March to 2-6 November. Topics include: model driven engineering for continuous architecting; component-based software engineering; architecture evaluation and quality aspects of software architectures; refactoring and evolving architecture design decisions and solutions; architecture frameworks and architecture description languages; linking architecture to requirements and/or implementation; reusable architectural solutions; software architecture for legacy systems and systems integration; architecting families of products; software architects roles and responsibilities; training, education, and certification of software architects;

	industrial experiments and case studies; technical debt management; software architecture design, evaluation, documentation, conformance, and reconstruction; etc.
November 08-13 (v)	28th ACM **Joint European Software Engineering Conference** and **Symposium on the Foundations of Software Engineering** (ESEC/FSE'2020). San Francisco, California, USA. Topics include: agile software development; component-based software engineering; configuration management and deployment; cyber physical systems; debugging; dependability, safety, and reliability; education; embedded software; emerging domains of software; empirical software engineering; formal methods; middleware, frameworks, and APIs; mining software engineering repositories; model-driven engineering; parallel, distributed, and concurrent systems; program analysis; program comprehension; program repair; programming languages; refactoring; reverse engineering; safety-critical systems; scientific computing; security, privacy and trust; software architecture; software economics and metrics; software evolution and maintenance; software modeling and design; software process; software product lines; software reuse; software testing; software visualization; specification and modeling languages; tools and environments; traceability; validation and verification; etc.
November 09-12 (v)	32nd **International Conference on Software Engineering Education and Training** (CSEET'2020). Munich, Germany. CSEET'2020 was first postponed from 28-31 July to 9-12 November, and later moved to a virtual event format. Topics include: Teaching formal methods (TFM), Teaching "real world" SE practices (TRW), Software quality assurance education (SQE), Global and distributed SE education (GDE), Open source in education (OSE), Cooperation between Industry and Academia (CIA), Training models in industry (TMI), Continuous education (CED), Methodological aspects of SE education (MAE), etc. Deadline for early registration: October 18, 2020.
November 09-13 (v)	23rd **Ibero-American Conference on Software Engineering** (CIbSE'2020), Curitiba, Brazil. CIbSE'2020 was first postponed from 4-8 May to 16-20 November, but then replaced by a virtual event on 9-13 November. Event includes Software Engineering Track (SET) and Experimental Software Engineering Track (ESELAW).
☺ November 15-17 (v)	34th **European Conference on Object-Oriented Programming** (ECOOP'2020). Berlin, Germany. ECOOP'2020 was moved from 13-17 July in Berlin, Germany, to 15-17 November in a virtual event format. Topics include: design, implementation, optimization, analysis, and theory of programs, programming languages, and programming environments.
☺ November 15-20 (v)	ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2020). Chicago, USA. Topics include: all aspects of software construction, at the intersection of programming, languages, and software engineering. Deadline for early registration: October 21, 2020.
	Nov 15-20 (v) — 13th ACM SIGPLAN **International Conference on Software Language Engineering** (SLE'2020). Topics include: areas ranging from theoretical and conceptual contributions, to tools, techniques, and frameworks in the domain of software language engineering; software language engineering rather than engineering a specific software language; software language design and implementation software language validation software language integration and composition software language maintenance (software language reuse, language evolution, language families and variability); domain-specific approaches for any aspects of SLE (design, implementation, validation, maintenance) empirical evaluation and experience reports of language engineering tools (user studies evaluating usability, performance benchmarks, industrial applications); etc.
☺ November 16-17 (v)	ACM SIGAda's **High Integrity Language Technology Workshop on Safe Languages and Technologies for Structured and Efficient Parallel and Distributed/Cloud Computing** (HILT'2020), Chicago, Illinois, USA. Co-located with SPLASH 2020. Sponsored by ACM SIGAda. Topics include: practical use of High Integrity languages, technologies, and methodologies in construction of safe, structured, highly parallel and/or distributed/cloud applications; safe and productive languages and frameworks for development of structured parallel and/or distributed applications (e.g. Rust, Concurrent Collections, Ada 202X, Parsl); practical tools for applying static analysis and formal methods to parallel and/or distributed/cloud applications (e.g. SPARKProver, Java Pathfinder); etc. Deadline for submissions: October 9, 2020 (nominations for SIGAda 2020 Awards).
November 16-20 (v)	16th **International Conference on integrated Formal Methods** (iFM'2020), Lugano, Switzerland. Topics include: recent research advances in the development of integrated approaches to formal modelling and analysis; all aspects of the design of integrated techniques, including language design,

	verification and validation, automated tool support and the use of such techniques in software engineering practice. Includes paper presentation on "Generating SPARK from Event-B Models". Deadline for registration: November 9, 2020.
November 18-20 (v)	27th **Static Analysis Symposium** (SAS'2020), Chicago, Illinois, USA. Topics include: all aspects of static analysis, such as abstract interpretation, data flow analysis, debugging, emerging applications, model checking, program optimizations and transformations, program verification, security analysis, tool environments and architectures, type checking, etc.
November 25-27 (v)	21st **International Conference on Product-Focused Software Process Improvement** (PROFES'2020). Turin, Italy. Topics include: experiences, ideas, innovations, as well as concerns related to professional software development and process improvement driven by product and service quality needs.
November 25-27 (v)	23rd **Brazilian Symposium on Formal Methods** (SBMF'2020), Ouro Preto, MG, Brazil. Topics include: applications of formal methods to software development, code generation testing, maintenance, evolution, reuse, ...; specification and modelling languages (such as logic and semantics for specification or/and programming languages, formal methods for timed, real-time, hybrid, or/and safety-critical systems, formal methods for cyber-physical systems, ...); theoretical foundations (such as type systems, models of concurrency, security, ...); verification and validation (such as abstraction, modularization or/and refinement techniques, static analysis, model checking, theorem proving, software certification, correctness by construction); experience reports on teaching formal methods, on industrial application of formal methods.
Nov 29 - Dec 03 (v)	18th **Asian Symposium on Programming Languages and Systems** (APLAS'2020), Fukuoka, Japan. Topics include: design of languages and type systems; domain-specific languages; compilers, interpreters, abstract machines; program analysis, verification, model-checking; software security; concurrency and parallelism; tools and environments for programming and implementation; etc.
Nov 30 - Dec 04 (v)	17th **International Colloquium on Theoretical Aspects of Computing** (ICTAC'2020), Macao S.A.R., China. Topics include: semantics of programming languages; theories of concurrency; theories of distributed computing; models of objects and components; timed, hybrid, embedded and cyber-physical systems; static analysis; software verification; software testing; model checking and automated theorem proving; interactive theorem proving; verified software, formalized programming theory; etc.
☺ December 01-04 (v)	41st IEEE **Real-Time Systems Symposium** (RTSS'2020). Houston, Texas, USA.
December 01-04 (h)	27th **Asia-Pacific Software Engineering Conference** (APSEC'2020), Singapore. Topics include: agile methodologies; component-based software engineering; configuration management and deployment; cyber-physical systems and Internet of Things; debugging and fault localization; embedded real-time systems; formal methods; middleware, frameworks, and APIs; model-driven and domain-specific engineering; open source development; parallel, distributed, and concurrent systems; programming languages and systems; refactoring; reverse engineering; security, reliability, and privacy; software architecture, modelling and design; software comprehension and traceability; software engineering education; software engineering tools and environments; software maintenance and evolution; software product-line engineering; software reuse; software repository mining; testing, verification, and validation; etc. Deadline for submissions: September 30 - October 15, 2020 (workshop papers), October 2, 2020 (posters).
December 02-04 (v)	19th **International Conference on Software Reuse** (ICSR'2020). Hammamet, Tunisia. ICSR'2020 was postponed from 9-11 November to 2-4 December, and moved to a virtual event format. Theme: "Reuse in emerging software engineering practices". Topics include: new and innovative research results and industrial experience reports dealing with all aspects of software reuse within the context of the modern software development landscape. Deadline for submissions: August 15, 2020 (paper abstracts), August 18, 2020 (full papers), October 2, 2020 (workshops, tutorials, Doctoral Symposium).
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

December 10-14 (v) — 18th **International Conference on High Performance Computing & Simulation** (HPCS'2020), Barcelona, Spain. HPCS'2020 was postponed from 26-30 October to 10-14 December, and moved to a virtual event format. Event includes: symposium on Formal Approaches to Parallel and Distributed Systems (4PAD'2020); workshops on Dependable and Resilient Many-Core and Exascale Computing (DRMEC 2020), Advances in Parallel Programming Models and Frameworks for the Multi-/Many-core Era (APPMM 2020), Modeling and Simulation of Parallel and Distributed Systems (MSPDS 2020); etc. Deadline for registration: October 28, 2020.

> Dec 10-14 (v) — 3rd **Special Session on Compiler Architecture, Design and Optimization** (CADO'2020). Topics include: integration of language features, representations, optimizations, and runtime support for parallelism; auto-parallelization; compiler-support for parallelism mapping, task scheduling, memory management, data distribution and synchronization, ...; compiler support for multi-core architectures, GPUs, CGRAs, FPGAs, and accelerators; code generation, optimization, synthesis and verification; platforms, tools, debuggers, and profilers; distributed computing and communication avoiding algorithms; etc.

December 11-13 — 14th **International Symposium on Theoretical Aspects of Software Engineering** (TASE'2020), Hangzhou City, China. TASE'2020 was postponed from 15-17 July to 11-13 December. Topics include: theoretical aspects of software engineering, such as abstract interpretation, component-based software engineering, cyber-physical systems, distributed and concurrent systems, embedded and real-time systems, formal verification and program semantics, integration of formal methods, language design, model checking and theorem proving, model-driven engineering, object-oriented systems, program analysis, reverse engineering and software maintenance, run-time verification and monitoring, software architectures and design, software testing and quality assurance, software safety, security and reliability, specification and verification, type systems, tools exploiting theoretical results, etc.

December 11-14 — 20th IEEE **International Conference on Software Quality, Reliability and Security** (QRS'2020), Macau, China. QRS'2020 was postponed from 27-31 July in Vilnius, Lithuania to 11-14 December in Macau, China. Topics include: reliability, security, availability, and safety of software systems; software testing, verification, and validation; program debugging and comprehension; fault tolerance for software reliability improvement; modeling, prediction, simulation, and evaluation; metrics, measurements, and analysis; software vulnerabilities; formal methods; operating system security and reliability; benchmark, tools, industrial applications, and empirical studies; etc.

☺ December 17-19 (v) — 18th IEEE **International Symposium on Parallel and Distributed Processing with Applications** (ISPA'2020), Exeter, UK. Deadline for early registration: November 17, 2020.

2021

January 18-20 (v) — 16th **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2021). Budapest, Hungary. Topics include: computer architecture, programming models, compilers and operating systems for embedded and general-purpose systems.

January 19-21 — 13th **Software Quality Days** (SWQD'2021), Vienna, Austria. Topics include: improvement of software development methods and processes, testing and quality assurance of software and software-intensive systems, domain specific quality issues (such as embedded, medical, automotive systems), novel trends in software quality, etc.

February 01-05 (v) — 19th **Australasian Symposium on Parallel and Distributed Computing** (AusPDC'2021), Australia. Topics include: all areas of parallel and distributed computing; multi-core systems; GPUs and other forms of special purpose processors; middleware and tools; parallel programming models, languages and compilers; runtime systems; resource scheduling and load balancing; reliability, security, privacy and dependability; etc. Deadline for submissions: October 18, 2020.

☺ Feb 27 - Mar 03 (v) — 26th ACM SIGPLAN **Symposium on Principles and Practice of Parallel Programming** (PPoPP'2021). Seoul, South Korea. Deadline for submissions: October 23, 2020 (workshops, tutorials).

Feb 27 - Mar 03 (v) — 30th ACM SIGPLAN 2021 **International Conference on Compiler Construction** (CC'2021), Seoul, South Korea. Co-located with CGO, HPCA, and PPoPP. Topics include: processing programs in the most general sense (analyzing, transforming or executing input that describes how a system operates, including traditional compiler construction as a special case); compilation and interpretation techniques (including program representation, analysis, and transformation; code generation, optimization, and synthesis; the verification thereof); run-time techniques (including memory management, virtual machines, and dynamic and just-in-time compilation); programming tools (including refactoring editors, checkers, verifiers, compilers, debuggers, and profilers); techniques, ranging from programming languages to micro-architectural support, for specific domains such as secure, parallel, distributed, embedded or mobile environments; design and implementation of novel language constructs, programming models, and domain-specific languages. Deadline for submissions: November 8, 2020 (abstracts), November 10, 2020 (full papers), January 5, 2021 (artifacts).

March 04-06

25th **International Conference on Engineering of Complex Computer Systems** (ICECCS'2020), Singapore. ICECCS'2020 was postponed from 28-31 October 2020 to 4-6 March 2021. Topics include: all areas related to complex computer-based systems, including the causes of complexity and means of avoiding, controlling, or coping with complexity, such as verification and validation, security and privacy of complex systems, model-driven development, reverse engineering and refactoring, software architecture, design by contract, agile methods, safety-critical and fault-tolerant architectures, real-time and embedded systems, systems of systems, cyber-physical systems and Internet of Things (IoT), tools and tool integration, industrial case studies, etc.

March 09-12 (v)

28th IEEE **Conference on Software Analysis, Evolution, and Reengineering** (SANER'2021), Honolulu, Hawaii, USA. Topics include: theory and practice of recovering information from existing software and systems; software analysis, parsing, and fact extraction of multi-language systems; mining software repositories; empirical studies in software re-engineering, maintenance, and evolution; software architecture evolution; software maintenance and re-engineering economics; software release engineering, continuous integration and delivery; evaluation and assessment of reverse engineering and re-engineering tools; software analysis and comprehension; education related issues; etc. Deadline for submissions: October 15, 2020 (main research track abstracts, workshop proposals), October 22, 2020 (main research track Papers), November 12, 2020 (ERA track abstracts, tool track and industry track abstracts, RENE track abstracts, late breaking ideas abstracts, late breaking ideas papers), November 19, 2020 (ERA track papers, tool track and industry track papers, RENE track papers), November 20, 2020 (journals first track papers).

March 10-12 (v)

29th **Euromicro International Conference on Parallel, Distributed and Network-Based Processing** (PDP'2021), Valladolid, Spain. Deadline for paper submissions: October 4, 2020.

March 22-26 (v)

36th ACM **Symposium on Applied Computing** (SAC'2021), Gwangju, Korea.

Mar 22-26 (v)

Software Verification and Testing Track (SVT'2021). Topics include: new results in formal verification and testing, technologies to improve the usability of formal methods in software engineering, applications of mechanical verification to large scale software, model checking, correct by construction development, model-based testing, software testing, static and dynamic analysis, abstract interpretation, analysis methods for dependable systems, software certification and proof carrying code, fault diagnosis and debugging, verification and validation of large scale software systems, real world applications and case studies applying software testing and verification, etc. Deadline for submissions: October 12, 2020 (regular papers, student research competition research abstracts).

Mar 22-26 (v)

Embedded Systems Track (EMBS'2021). Topics include: the application of both novel and well-known techniques to the embedded systems development. Deadline for submissions: October 12, 2020 (full papers).

☺ Mar 22-26 (v)

Track on Programming Languages (PL'2021). Topics include: technical ideas and experiences relating to implementation and application of programming languages, such as compiling techniques, domain-specific languages, garbage collection, language design and implementation, languages for modeling, model-driven development, new programming language ideas and concepts, practical experiences with programming languages, program analysis and verification, etc.

Mar 22-26 (v)

16th Track on Dependable, Adaptive, and Secure Distributed Systems (DADS'2021). Topics include: Dependable, Adaptive, and secure Distributed Systems (DADS); modeling, design, and engineering of DADS; foundations and formal methods for DADS; etc.

March 27 (h) — IEEE **International Conference on Code Quality** (ICCQ'2021), Moscow, Russia. Topics include: static analysis, program verification, bug detection, and software maintenance. Deadline for submissions: December 4, 2020 (papers).

March 27 - April 1 (h) — 24th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2021), Luxembourg, Luxembourg. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), TACAS (Tools and Algorithms for the Construction and Analysis of Systems), SV-COMP (the 10th Competition on Software Verification). Deadline for submissions: October 15, 2020 (papers).

☺ April 07-09 (h) — 29th **International Conference on Real-Time Networks and Systems** (RTNS'2021), Nantes, France. Topics include: real-time applications design and evaluation (automotive, avionics, space, railways, telecommunications, process control, multimedia), real-time aspects of emerging smart systems (cyber-physical systems and emerging applications, ...), real-time system design and analysis (real-time tasks modeling, task/message scheduling, mixed-criticality systems, Worst-Case Execution Time (WCET) analysis, security, ...), software technologies for real-time systems (model-driven engineering, programming languages, compilers, WCET-aware compilation and parallelization strategies, middleware, Real-time Operating Systems (RTOS), ...), formal specification and verification, real-time distributed systems, etc. Deadline for submissions: November 27, 2020.

April 12-15 (h) — 27th **International Working Conference on Requirements Engineering: Foundation for Software Quality** (REFSQ'2021), Essen, Germany. Deadline for submissions: November 9, 2020 (research paper abstracts), November 16, 2020 (research papers).

April 12-16 (v) — 14th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2021), Porto de Galinhas, Brazil. Topics include: manual testing practices and techniques, security testing, model based testing, test automation, static analysis and symbolic execution, formal verification and model checking, software reliability, testability and design, testing and development processes, testing in specific domains (such as embedded, concurrent, distributed, ..., and real-time systems), testing/debugging tools, empirical studies, experience reports, etc. Deadline for submissions: October 5, 2020 (abstracts), October 12, 2020 (papers), December 10, 2020 (journal-first presentations).

April 17-23 (h) — 12th ACM/SPEC **International Conference on Performance Engineering** (ICPE'2021), Rennes, France. Deadline for submissions: October 16, 2020 (research abstracts, industrial/experience abstracts, workshops), October 23, 2020 (research papers, industrial/experience papers), December 14, 2020 (artifact registration), December 21, 2020 (artifact submission), January 20, 2021 (posters, demos, tutorials, work-in-progress papers).

May 18-21 — 14th **Cyber-Physical Systems and Internet of Things Week** (CPS Week'2021), Nashville, Tennessee, USA. Event includes: 5 top conferences, HSCC, ICCPS, IPSN, RTAS, and IoTDI, multiple workshops, tutorials, competitions and various exhibitions from both industry and academia. Deadline for submissions: October 26, 2020 (papers).

> ☺ May 18-21 — 27th IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS'2021). Topics include: systems research related to embedded systems and time-sensitive systems, ranging from traditional hard real-time systems to embedded systems without explicit timing requirements; papers describing original systems, applications, case studies, methodologies, and algorithms that contribute to the state of practice in design, implementation, verification, and validation of embedded systems or time-sensitive systems. Deadline for submissions: October 26, 2020 (papers)..

May 19-21 (h) — 9th **International Conference on Fundamentals of Software Engineering** (FSEN'2021), Tehran, Iran. Topics include: all aspects of formal methods, especially those related to advancing the application of formal methods in the software industry and promoting their integration with practical engineering techniques; software specification, validation, and verification; software architectures and their description languages; integration of formal and informal methods; component-based systems; cyber-physical software systems; model checking and theorem proving; software verification; CASE tools and tool integration; industrial applications; etc. Deadline for submissions: October 18, 2020 (abstracts), October 30, 2020 (papers).

May 23-29	43rd **International Conference on Software Engineering** (CSE'2021), Madrid, Spain. Topics include: the full spectrum of Software Engineering, such as testing and analysis (software testing, program analysis, validation and verification, fault localization, formal methods, programming languages), empirical software engineering (mining software repositories, software ecosystems, ...), software evolution (evolution and maintenance, debugging, program comprehension, API design and evolution, configuration management, release engineering and DevOps, software reuse, refactoring, reverse engineering, ...), social aspects of software engineering (human aspects of software engineering, agile methods and software processes, software economics, ethics in software engineering, ...), requirements, modeling, and design (requirements engineering, modeling and model-driven engineering, software architecture and design, tools and environments, variability and product lines, parallel, distributed, and concurrent systems, ...), dependability (software security, privacy, reliability and safety, performance, embedded / cyber-physical systems, ...), etc. Deadline for submissions: October 1, 2020 (IEEE TCSE Harlan Mills Award nominations).
May 24-28 (h)	13th **NASA Formal Methods Symposium** (NFM'2021), Norfolk, Virginia, USA. Topics include: challenges and solutions for achieving assurance for critical systems; formal verification, model checking, and static analysis techniques; theorem proving; techniques and algorithms for scaling formal methods; design for verification and correct-by-design techniques; experience report of application of formal methods in industry; use of formal methods in education; applications of formal methods in the development of autonomous systems, safety-critical systems, concurrent and distributed systems, cyber-physical, embedded, and hybrid systems, ...; etc. Deadline for submissions: November 27, 2020 (abstracts), December 4, 2020 (papers).
♦ June 07-11	25th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC 2021 aka Ada-Europe 2021). Santander, Spain. AEiC'2020 was postponed from 8-12 June 2020 to 7-11 June 2021. Sponsored by Ada-Europe.
☺ July 12-16	35th **European Conference on Object-Oriented Programming** (ECOOP'2021), Aarhus, Denmark. Topics include: design, implementation, optimization, analysis, testing, verification, and theory of programs, programming languages, and programming environments. Deadline for submissions: October 13, 2020 (ACM Transactions on Programming Languages and Systems journal first papers), November 2, 2020 (Science of Computer Programming journal first papers), January 11, 2021 (papers).
October 10-15	**Embedded Systems Week** 2021 (ESWEEK'2021). Shanghai, China. The venues for ESWEEK 2020 and 2021 were swapped. ESWEEK 2020 was held in Hamburg, Germany from September 20-25, 2020, and ESWEEK 2021 will be held in Shanghai, China from October 10-15, 2021. Includes CASES'2021 (International Conference on Compilers, Architectures, and Synthesis for Embedded Systems), CODES+ISSS'2021 (International Conference on Hardware/Software Codesign and System Synthesis), EMSOFT'2021 (International Conference on Embedded Software).
November 20-26	24th **International Symposium on Formal Methods** (FM'2021), Beijing, China. Topics include: formal methods in a wide range of domains including software, computer-based systems, systems-of-systems, cyber-physical systems, security, human-computer interaction, manufacturing, sustainability, energy, transport, smart cities, and healthcare; formal methods in practice (industrial applications of formal methods, experience with formal methods in industry, tool usage reports, experiments with challenge problems); tools for formal methods (advances in automated verification, model checking, and testing with formal methods, tools integration, environments for formal methods, and experimental validation of tools); formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, and method integration); etc. Deadline for submissions: April 30, 2021 (abstracts), May 6, 2021 (full papers).
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

25th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2021)

7-11 June 2021, Santander, Spain

(C) RMR

Conference Chair
Michael González Harbour
Universidad de Cantabria, Spain
mgh@unican.es

Program Chairs
Mario Aldea Rivas
Universidad de Cantabria, Spain
aldeam@unican.es
J. Javier Gutiérrez García
Universidad de Cantabria, Spain
gutierjj@unican.es

Work-in-Progress Chair
Kristoffer Nyborg Gregertsen
SINTEF Digital, Norway
kristoffer.gregertsen@sintef.no

Tutorial & Workshop Chair
Jorge Garrido Balaguer
Universidad Politécnica de Madrid, Spain
jorge.garrido@upm.es

Industrial Chair
Patricia Balbastre Betoret
Universitat Politècnica de València, Spain
patricia@ai2.upv.es

Exhibition & Sponsorship Chair
Ahlan Marriott
White Elephant GmbH, Switzerland
software@white-elephant.ch

Publicity Chair
Dirk Craeynest
Ada-Belgium & KU Leuven, Belgium
dirk.craeynest@cs.kuleuven.be

Ada europe

In cooperation with

acm In-Cooperation

SIGAda, SIGPLAN, SIGBED
(approval pending)

and

Ada Resource ARA Association

General Information

The **25th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2021 aka Ada-Europe 2021)** will take place in Santander, Spain. The conference schedule includes a technical program, vendor exhibition and parallel tutorials and workshops.

Despite the COVID-19 situation which led to the cancellation of the previous edition of the conference, there is a firm commitment to celebrate the 2021 edition in any case. The initial goal is to have a mixed model with both in-person and remote participation. If the situation so requires it, the conference would be held as a full virtual event.

The 2021 edition of the conference continues the major revamp of the in-person registration fees introduced in 2019, redesigned to extend participation from industry and academia, and to reward contributors, especially but not solely, students and post-doc researchers.

Schedule

7 January 2021	Submission of journal-track papers, industrial presentation outlines, and tutorial and workshop proposals
19 March 2021	Notification of journal-track paper presentations, industrial presentations, tutorials and workshops
31 March 2021	Submission of Work-in-Progress (WiP) papers and Invited Presentation proposals
30 April 2021	Notification of acceptance for WiP papers and Invited Presentations

Topics

The conference is a leading international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a challenging variety of application domains. The program will allow ample time for keynotes, Q&A sessions and discussions, and social events. Participants include practitioners and researchers from industry, academia and government organizations active in the promotion and development of reliable software technologies. Should the situation allow it, the presenters will be given the choice of in-person or virtual participation.

The topics of interest for the conference include but are not limited to:

- **Design and Implementation of Real-Time and Embedded Systems,**
- **Design and Implementation of Mixed-Criticality Systems,**
- **Theory and Practice of High-Integrity Systems,**
- **Software Architectures for Reliable Systems,**
- **Methods and Techniques for Quality Software Development and Maintenance,**
- **Ada Language and Technologies,**
- **Mainstream and Emerging Applications with Reliability Requirements,**
- **Achieving and Assuring Safety in Machine Learning Systems,**
- **Experience Reports on Reliable System Development,**
- **Experiences with Ada.**

Refer to the conference website for the full list of topics.

www.ada-europe.org/conference2021

(C) Pachi Hondal

Call for Journal-Track Papers

The **journal-track papers** submitted to the conference are full-length papers that must describe mature research work on the conference topics. They must be original and shall undergo anonymous peer review.

Accepted journal-track papers will get a presentation slot within a technical session of the conference and they will be published in an open-access special issue of the **Journal of Systems Architecture** (Q2 in the JCR and SJR ranks) with no additional costs to authors. The corresponding authors shall submit their work by 7 January 2021 via the Special Issue web page: *https://www.journals.elsevier.com/journal-of-systems-architecture/call-for-papers/special-issue-on-reliable-software-technologies-aeic2021*.

Submitted papers must follow the guidelines provided in the "Guide-for-Authors" of the JSA (*https://www.elsevier.com/journals/journal-of-systems-architecture/1383-7621/guide-for-authors*). In particular, JSA does not impose any restriction on the format or extension of the submissions.

Call for WiP-Track Papers

The **Work-in-Progress papers (WiP-track)** are short (4-page) papers describing evolving and early-stage ideas or new research directions. They must be original and shall undergo anonymous peer review. The corresponding authors shall submit their work by 31 March 2021, via *https://easychair.org/conferences/?conf=aeic2021*, strictly in PDF and following the Ada User Journal style (*http://www.ada-europe.org/auj/*).

Authors of accepted WiP-track papers will get a presentation slot within a regular technical session of the conference and will also be requested to present a poster. The papers will be published in the Ada User Journal as part of the proceedings of the Conference.

The conference is listed in the principal citation databases, including DBLP, Scopus, Web of Science, and Google Scholar. The Ada User Journal is indexed by Scopus and by EBSCOhost in the Academic Search Ultimate database.

Call for Industrial Presentations

The conference seeks **industrial presentations** that deliver insightful information value but may not sustain the strictness of the review process required for regular papers. The authors of industrial presentations shall submit their proposals, in the form of a short (one or two pages) abstract, by 7 January 2021, via *https://easychair.org/conferences/?conf=aeic2021*, strictly in PDF and following the Ada User Journal style (*http://www.ada-europe.org/auj/*).

The Industrial Committee will review the submissions anonymously and make recommendations for acceptance. The abstract of the accepted contributions will be included in the conference booklet, and authors will get a presentation slot within a regular technical session of the conference.

These authors will also be invited to expand their contributions into articles for publication in the Ada User Journal, as part of the proceedings of the Industrial Program of the Conference.

Awards

Ada-Europe will offer an honorary **award for the best presentation**. All journal-track and industrial presentations are eligible.

Call for Invited Presentations

The **invited presentations** are intended to allow researchers to present paramount research results that are relevant to the conference attendees. There will be no publication associated to these presentations, which may include previously published works, relevant new tools, methods or techniques.

The invited presentations will be allocated a presentation slot. Presentations can be delivered remotely to facilitate the participation.

The Program Committee will select invited presentation proposals that may be submitted by e-mail to one of the Program Chairs as a one-page summary of the proposed presentation, along with the information and/or links required to show the relevance of the covered topic.

Call for Educational Tutorials

The conference is seeking tutorials in the form of educational seminars including hands-on or practical demonstrations. Proposed tutorials can be from any part of the reliable software domain, they may be purely academic or from an industrial base making use of tools used in current software development environments. We are also interested in contemporary software topics, such as IoT and artificial intelligence and their application to reliability and safety.

Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half day or full day), and the intended level of the tutorial (introductory, intermediate, or advanced). All proposals should be submitted by e-mail to the Educational Tutorial Chair.

The authors of accepted full-day tutorials will receive a complimentary conference registration. For half-day tutorials, this benefit is halved. The Ada User Journal will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

Workshops on themes that fall within the conference scope may be proposed. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference days. Workshop proposals should be submitted by e-mail to the Workshop Chair. The workshop organizer shall also commit to producing the proceedings of the event, for publication in the Ada User Journal.

Call for Exhibitors

The commercial exhibition will span the core days of the main conference. As an alternative to the traditional physical exhibition, virtual exhibition activities will be possible. Vendors and providers of software products and services should contact the Exhibition Chair for information and for allowing suitable planning of the exhibition space and time.

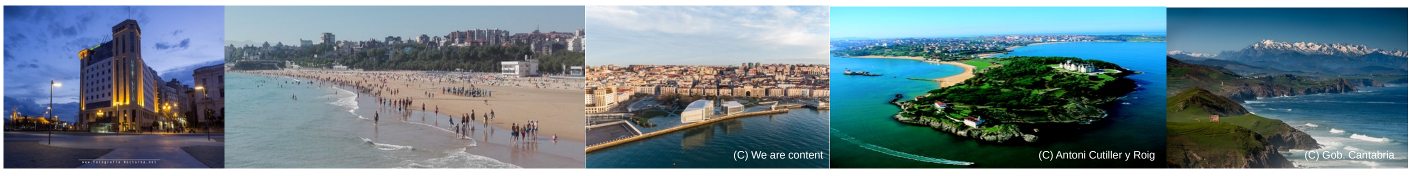
Special Registration Fees

Authors of accepted contributions and all students will enjoy reduced registration fees. In addition, there will be low registration fees for virtual participants.

Venue

Santander is a nice tourist city in the north of Spain, with a well-connected airport and at a 100 km drive from Bilbao airport.

The conference venue and hotel is the Bahia Hotel in the city center and beside Santander bay.



Ada User Journal

Call for Contributions

Topics: **Ada, Programming Languages**, **Software Engineering Issues** and **Reliable Software Technologies** in general.

Contributions: **Refereed Original Articles**, **Invited Papers**, **Proceedings** of workshops and panels and **News and Information** on Ada and reliable software technologies.

More information available on the
Journal web page at

http://www.ada-europe.org/**auj**

Online archive of past issues at http://www.ada-europe.org/auj/archive/

An Overview of Ada 202x

Jeff Cousins CEng FIET

Member and former chair of the Ada Rapporteur; email: jeffrey.cousins@btinternet.com

1 Introduction

Let us begin with why do we need another revision, when Ada has been used to successfully deliver so many projects around the world, indeed across the solar system?

The world has moved on. The most significant external factor has probably been the growth of the number of cores on a processor. Making use of the first multi-core processors was relatively easy for Ada compared with other languages as tasking had been included from the outset. Maybe latent timing problems would emerge when tasks were truly running on different processors instead of just pretending to, but these problems were small scale compared with, say, elaboration order problems when moving between different vendors' compilers.

These days a processor may have dozens of cores, maybe in the future it will be hundreds. Thus the first improvement given in the list of instructions (see below) from WG 9 (the ISO/IEC Working Group responsible for Ada), to the Ada Rapporteur Group (ARG), was finer grained control of parallelism.

The ARG follows the following instructions from WG 9, extracted from ISO/IEC JTC 1/SC 22/WG 9 N571:

"The ARG is requested to pay particular attention to the following two categories of improvements:

A. Improvements that will maintain or improve Ada's advantages, especially in those user domains where safety and security are prime concerns;

B. Improvements that will remedy shortcomings in Ada.

Improvements of special interest in these categories are:

- Improving the capabilities of Ada on multi-core and multi-threaded architectures;
- Improving the ability to write and enforce contracts for Ada entities (for instance, via preconditions);
- Improving the use and functionality of the predefined containers;
- Improving support for Unicode in the language and predefined libraries.

These are all examples of improvements in category A, except for the last which is an example of an improvement in category B."

New real-time features have also arisen, primarily from recommendations from the series of International Real-Time Ada Workshops (IRTAW).

As always, there is the balancing act to be struck between keeping the language stable and backwardly compatible, and adding new features to keep with the times.

Thanks again to John Barnes for his review comments.

Notes:
1) The alternative number is only given for those AIs with more than one alternative.
2) Contemporary English is used where possible, but where a word is quoted from the RM, then of necessity the US spelling is used.

2 Parallelism

Although parallelism may be the first improvement on the list, the parallelism features may be the hardest of the new features to add, so unfortunately they may be the last ones to be implemented.

OpenMP is a promising platform to use for the parallelism features though. It already provides facilities for C, C++ and Fortran. A logical thread of control could be mapped on to an OpenMP lightweight thread. OpenMP has made similar design decisions to Ada 202x. For example, if either an exception is propagated out of a logical thread of control, or there is an explicit transfer of control out of a logical thread of control, then an attempt is made to cancel all logical threads of control within the construct.

Some of the new Ada features are clearly inspired by existing SPARK 2014 features, especially *Global-in and global-out annotations (AI12-0079-3)*, but generalising these to cover the whole of the Ada language, not just the SPARK subset, has been far from trivial.

2.1 Parallel constructs and the need to consider potential blocks and conflicts over the access to global data

Parallel operations (AI12-0119) is the prime AI for satisfying the first instruction from WG 9 to the ARG, i.e. "Improving the capabilities of Ada on multi-core and multi-threaded architectures". Two parallel constructs are provided, namely parallel blocks and parallel loops.

A parallel block consists of a set of concurrent activities each specified by a handled sequence of statements, separated by the reserved word **and**, analogous to the syntax for a select statement where the alternatives are separated by the reserved word **or**.
A parallel loop defines a loop body which is designed such that the various iterations of the loop can run concurrently. The implementation is expected to group the iterations into "chunks" to avoid creating an excessive number of physical threads of control, but each iteration is nevertheless

considered for most purposes as its own separate logical thread of control.

Both constructs start with the new reserved word **parallel** to clearly indicate that these constructs are designed for parallel execution. The implementation might still not execute the constructs in parallel, but the intent is that if multiple processors are available, some or all of them should be allocated to the execution of the construct.
An example of using parallel blocks when searching a binary tree:

```
type Expression is tagged null record;
  -- Components will be added by each extension
type Expr_Ptr is access all Expression'Class;
type Binary_Operation is new Expression with
  record
  -- An internal node in an Expression tree
    Left, Right : Expr_Ptr;
  end record;
procedure Traverse (T : Expr_Ptr) is
begin
  -- Recurse down the binary tree
  if T /= null and then
                T.all in Binary_Operation'Class then

    parallel do
      Traverse (T.Left);
    and
      Traverse (T.Right);
    and
      Ada.Text_IO.Put_Line ("Processing " &
                 Ada.Tags.Expanded_Name (T'Tag));
    end do;

  end if;
end Traverse;
```

An example of using parallel blocks when searching a string for a particular character:

```
function Search (S : String;
                 Char : Character) return Boolean is
begin
  if S'Length <= 1000 then
    -- Sequential scan
    return (for some C of S => C = Char);
  else
    -- Parallel divide and conquer
    declare
      Mid : constant Positive := S'First + S'Length/2 – 1;
    begin

      parallel do
        for C of S(S'First .. Mid) loop
          if C = Char then
            return True;   -- Terminates enclosing "do"
          end if;
        end loop;
      and
        for C of S(Mid + 1 .. S'Last) loop
          if C = Char then
            return True;   -- Terminates enclosing "do"
          end if;
        end loop;
      end do;

      -- Not found
      return False;
    end;
  end if;
end Search;
```

An example of using a parallel loop when initialising a two-dimensional Boolean array:

```
parallel for I in Grid'Range(1) loop
  Grid(I, 1) := (for all J in Grid'Range(2) =>
                                    Grid(I,J) = True);
end loop;
```

It is intended that the parallel constructs use lightweight threading so as to incur fewer overheads than tasking. To reduce implementation complexity and reduce the risk of deadlock, blocking is not allowed in a parallel construct, thus it is a bounded error to invoke an operation that is potentially blocking during the execution of a parallel construct. The compiler may complain if a parallel sequence calls a potentially blocking operation. It may also complain if parallel sequences have conflicting global side-effects.

Whereas *Parallel operations (AI12-0119)* provides the mechanism for iterating in parallel over the elements of an array, *Parallel Container Iterators (AI12-0266)* provides the equivalent mechanism for iterating over containers. The optional reserved word **parallel** may be placed before a **for** statement for its iterator form, not just its loop parameter form. The interfaces in Ada.Containers.Iterator_Interfaces are extended to include two new interfaces, a parallel iterator interface, and a reversible parallel iterator interface.

The **of** form of a parallel loop should be similar whether it is an array (which can be multi-dimensional) or a container being iterated over, e.g. for a multi-dimensional array:

```
type Matrix is array
        (Integer range <>, Integer range <>) of Real;
Board : Matrix (1 .. 8, 1 .. 8);
parallel for Element of Board loop
  Element := Element * 2.0;
  -- Double each element of the two-dimensional array
  -- Board
end loop;
```

2.2 Nonblocking and data race checks

Data race and non-blocking checks for parallel constructs (AI12-0267), amended by *Revise the conflict check policies to ensure compatibility (AI12-0298),* provides the rules to check for blocking operations and for race conditions within parallel constructs. A "data race" occurs when two concurrent activities attempt to access the same data object without appropriate synchronization and at least one of the accesses updates the object. Such "conflicting" concurrent activities are considered erroneous. The first AI introduces the notion of Conflict Check policies, to control the degree of checking for potential data races.

It is important that the default for the new parallel constructs is that all possible conflicts are checked, but for backward compatibility we want the default for tasking constructs to be no checks. Thus the pragma Conflict_Check_Policy permits two separate policies, one for parallel constructs, and one for tasking. The policies for parallel constructs include:

- No_Parallel_Conflict_Checks
- Known_Parallel_Conflict_Checks
- All_Parallel_Conflict_Checks

and similarly the policies for tasking include:

- No_Tasking_Conflict_Checks
- Known_Tasking_Conflict_Checks
- All_Tasking_Conflict_Checks.

The default policy is:

```
pragma Conflict_Check_Policy
                    (All_Parallel_Conflict_Checks,
                     No_Tasking_Conflict_Checks);
```

If the checking policies for parallel constructs and tasking are the same then the pragma may just take one parameter, thus:

```
pragma Conflict_Check_Policy (No_Conflict_Checks);
```

which is shorthand for:

```
pragma Conflict_Check_Policy
                    (No_Parallel_Conflict_Checks,
                     No_Tasking_Conflict_Checks);
```

Similarly for the policies Known_Conflict_Checks and All_Conflict_Checks.

2.3 Defining Nonblocking

Nonblocking subprograms (AI12-0064-2) adds the aspect Nonblocking and the attribute Nonblocking to Ada. These allow specifying and querying the blocking status of a subprogram. If a subprogram is declared to be non-blocking, the Ada compiler will attempt to verify that it does not execute any potentially blocking operations. Potentially blocking operations include **select**, **accept**, **entry** call**,** **delay** and **abort** statements, creating or activating a task, or calling something else which in turn calls one of these.

The deadlock case may not be detectable at compile time though, in which case the runtime check of pragma Detect_Blocking can be used.

Note that the Nonblocking aspect may be specified for generic formal parameters.

Rather than specify the Nonblocking aspect for many individual subprograms, it can be specified for a **package**, **protected type**, **task** or **generic**, in which case it sets the default for everything within. Indeed for a **protected type**, it may not be specified for individual operations within.

The Nonblocking aspect is fixed as False for an **entry**, and as True for a predefined operator of an elementary type, as one might expect.

Contracts for container operations (AI12-0112) then uses the Nonblocking aspect to specify the non-blocking status for the predefined Containers.

Specifying Nonblocking for Language-Defined Units (AI12-0241) then uses the Nonblocking aspect to specify the non-blocking status for the remainder of Ada's own units (that is, child units of packages Ada and System).

Nonblocking for Unchecked_Deallocation is wrong (AI12-0319) tidies up some of the details of the Nonblocking aspect. An object coming into or out of existence is not always a passive affair: if it is of a controlled type, then it may have Initialize, Adjust and Finalize procedures; there might be default initialisation of record components, which could call a function; and if on the heap there will be Allocate, Deallocate, and Storage_Size subprograms. For a type to be non-blocking, whichever of these are applicable need to be non-blocking too. (Default initialisation of scalars is not of concern as the Default_Value aspect may only have a static value.)

The standard storage pool(s) are defined to be non-blocking. For a user-defined storage pool to be non-blocking, its Allocate, Deallocate, and Storage_Size subprograms must also be non-blocking. The Unchecked_Deallocation generic invokes a Finalize procedure, so for an instance of it to be non-blocking it must be instantiated with a non-blocking type.

Fixes for Nonblocking (AI12-0374-2) clarifies what happens for generic instantiations. At the point of instantiation, the Nonblocking aspects of the actual generic parameters are "and"-ed with the Nonblocking aspects of the operations within the generic. Thus a non-blocking generic can be instantiated with blocking actuals, in which case the instance will allow blocking. If the instance is required to be non-blocking, then the specific instance can be declared as such.

Also, the Nonblocking aspect may be specified on subtypes, not just types, so that predicates on some subtypes of a given type may call a blocking operation and predicates on some other subtypes of the type may not.

The Nonblocking aspect should also account for preconditions, postconditions, predicates and type invariants applicable to the call of the subprogram.

Fixups for Global annotations (AI12-0380) (which, despite the name, is mostly applicable to both the Nonblocking aspect and the Global annotations) provides finer grain control regarding the use of generic formal parameters and dispatching calls, in optional Annex H for High Integrity Systems.

Normally entities declared within a generic unit are presumed to make use of all the generic formal parameters. This AI adds aspect Use_Formal followed by a list of which generic formal parameters are actually used (enclosed by round brackets and comma-separated if more than one), the reserved word **null** for none, or the reserved word **all** for all of them (which is the default anyway).

Normally dispatching calls are checked using the applicable Global'Class aspects. This AI adds aspect Dispatching followed a list of dispatching calls (enclosed by round brackets and comma-separated if more than one, and each followed by the name of an object also enclosed by round brackets) that may potentially be called, for which the caller of the original subprogram will account for any globals accessed.

For example:

```
type T is tagged private
  with Input => Stream_Input;
procedure Fill (X : out T'Class;
                Str : aliased in out
                  Ada.Streams.Root_Stream_Type'Class)
  with Global => in Debug,
       Dispatching =>
                    (T'Input (X), Display (X), Read (Str));
.. -- That was the spec of Fill; the body is below
procedure Fill (X : out T'Class;
                Str : aliased in out
               Ada.Streams.Root_Stream_Type'Class) is
begin
  X := T'Input (Str'Access);
  if Debug then
    Display (X);
  end if;
end Fill;
```

2.4 Defining access to global data

Global-in and global-out annotations (AI12-0079-3) allow the programmer to specify what global data a subprogram uses, in a manner that is similar to that by which subprogram parameters are specified. Specifying the "side effects" (i.e. effects other than via a parameter) of a subprogram makes it easier for static analysis tools to reason. For example:

```
type Operating_Mode_Type is
      (Initialising, Normal, Fallback, Shutting_Down);
type Status_Type is (Success, Inaccurate, Failed);

Data_Table       : …;
Operating_Mode : Operating_Mode_Type;
Status             : Status_Type;

procedure Process_Data_Table
  with
    Global => (in     => Operating_Mode;
               out => Status;
            in out => Data_Table);
```

This should be fairly familiar to SPARK 2005 users, since as we are extending the language proper, we may revert to using the reserved words **in**, **out** and **in out** rather than the SPARK 2014 terms Input, Output and In_Out.

The Global'Class aspect can be specified for a dispatching subprogram, giving an upper bound on the set of global variables that any subprogram dispatched to may access.

For each mode there can be a list of global variables (comma-separated if more than one), the reserved word **all** for all global variables, or the reserved word **synchronized** for all synchronized variables (i.e. tasks, protected objects and atomic objects, the implication being that accesses to them are thread-safe). (*Meaning of Global when there is no mode (AI12-0375)* tweaks the syntax to use semicolons to separate the list of variables for each mode as in the examples above).

The intention is that although advanced users may impose stricter requirement on themselves, the typical user should have to specify few, if any, global aspects. Thus the global aspect for a library unit usually defaults to "Unspecified", i.e. read and write of an unspecified set of global variables (sounds a bit like Rumsfeld's "known unknowns"!), although to **null** for Pure library units, i.e. no read or write of any global variable. For other entities, the global aspect defaults to that of the enclosing library unit.

Besides covering any global variables accessed by the body of the subprogram, the global aspect should also cover those accessed by any preconditions, postconditions, predicates and type invariants. Global variables accessed by other subprograms that the subprogram calls should normally also be identified, though if the other subprogram is passed in as an access-to-subprogram parameter then it is up to the caller of the original subprogram to take account of the effects of whatever subprogram it passes in. If an access-to-variable value is created then presumably the variable that it designates is going to be written, and if an access-to-constant value is created then presumably the constant that it designates is going to be read, so these accesses should be identified too. However, the core language does not check accesses to objects reached via dereferences of access values, or via a generic formal parameter.

Optional Annex H, for High Integrity Systems, adds restriction No_Unspecified_Globals, disallowing the Global and Global'Class for a library-level entity from being set or defaulting to Unspecified, thereby forcing the specification of Global. It also adds the restriction No_Hidden_ Indirect_Globals, requiring that any accesses to objects reached via dereferences of access values are identified. For example:

```
package P is
  type G is private;
  type Ref (Data : access T) is null record;
  Glob : G;
  ...
  function F (C : aliased in out Container;
              Pos : Cursor) return Ref
    with Global => in Glob;
  ...
private
  type G is record
    Info : access T;
  end record;
end P;
```

```
package body P is
  ...
  function F (C : aliased in out Container;
              Pos : Cursor) return Ref is
  begin
    return Ref'(Data => Glob.Info);
      -- Error!
      -- The above returns a writable reference to
      -- Glob.Info.all, but Glob is of mode in, and
      -- Glob.Info.all is reachable from Glob.
  end F;
  ...
end P;
```

Optional Annex H allows the global aspect to be specified for subtypes and certain generic formal parameters.

Optional Annex H also provides an extension for dealing with "handles", for example the File_Type of Text_IO or the Generator of Discrete_Random. Hence, in:

```
procedure Put (File : in File_Type; Item : in String);
```

the File parameter is of mode **in** as the parameter itself isn't modified, yet the state associated with the file is modified. This can now be indicated using an overriding global mode, thus:

```
procedure Put (File : in File_Type; Item : in String)
  with Global => overriding in out File;
```

Fixups for Global annotations (AI12-0380) provides finer grain control regarding the use of generic formal parameters and dispatching calls, in optional Annex H. See the preceding section on Defining Nonblocking for details.

Contracts for container operations (AI12-0112) then uses the global annotations mechanism to specify the Global aspect for the predefined Containers.

Default Global aspect for language-defined units (AI12-0302) then uses the global annotations mechanism to specify the Global aspect for the remainder of Ada's own units (that is, child units of packages Ada and System).

For most language-defined packages (that are not Pure) an explicit value of "**synchronized in out** <unit_name>" (i.e. read and write of the set of global variables that are tasks, protected objects, or atomic objects, of the containing package) is added.

But where some unknown, unsynchronised variable holds state (such as Current_Input or Current_Output for Text_IO) then only "**in out** <unit_name>" can be stated. This would mean that two concurrent subprogram calls using either Current_Input or Current_Output would be considered to conflict.

Some parameters may be "handles", for example the File_Type of Text_IO, which even if of mode **in** may be used by a subprogram to update state. For these the value will be "**overriding in out** <param>".

2.5 Reduction

Reduction expressions are another new form of expression, added by Ada 202x, on top of those already added by Ada 2012 (e.g. if expressions, case expressions, quantified expressions). Before the parallel forms can be described, the sequential forms have to be described. These make use of *Container aggregates; generalized array aggregates (AI12-0212)* (see the Containers and Iterators section), which in turn makes use of *Index parameters in array aggregates (AI12-0061)* (see the Others section), so first a preview of them.

Index parameters in array aggregates (AI12-0061) allows a loop parameter to be used in an array aggregate, for example:

```
subtype Index_Type is Positive range 1 .. 10;
type Array_Type is array (Index_Type) of Positive;
Squares_Array : Array_Type :=
                        (for I in Index_Type => I * I);
```

Container aggregates; generalized array aggregates (AI12-0212) introduces container aggregates for initialising containers, using square brackets not round brackets (parentheses), and allowing square brackets as an alternative to round brackets for array aggregates.

Iteration is possible within the container aggregate, for example to create a set whose elements all have double the value of the corresponding elements of another set:

```
Doubles_Set : My_Set := [ for Item of X => Item * 2 ];
```

Map-Reduce attribute (AI12-0262) provides a mechanism to take a stream of values – a "value sequence"- from an aggregate, and repeatedly apply the same operation to combine the values to produce a single result. Examples include adding a sequence of squares for "sum of squares" or multiplying a sequence of numbers when calculating a factorial. An initial value is required, usually something neutral that has no effect on the result, such as 0 for addition or 1 for multiplication. A parallel version is provided, though if the combining operation is something simple such as addition then the overhead of managing the parallelism is likely to outweigh any performance benefit of performing the additions in parallel. Some examples:

```
-- A reduction expression that outputs the
-- sum of squares
Put_Line ("Sum of Squares is" & Integer'Image
  ([ for I in 1 .. 10 => I**2 ]'Reduce("+", 0));
-- An expression function that returns its result as
-- a Reduction Expression
function Factorial (N : Natural) return Natural is
  ([ parallel for J in 1..N => J ]'Reduce("*", 1));
```

It is important to note that the values are not put in some temporary array then combined, but are combined "on the fly" as each value is produced.

Shorthand Reduction Expressions for Objects (AI12-0242) provides a shorthand for cases where the object is an array or iterable container. For example:

```
Sum : constant Integer := A'Reduce("+", 0);
```

is short for:

```
Sum : constant Integer :=
    [ for Value of A => Value ]'Reduce("+", 0);
```

Similarly:

```
Sum : constant Integer := A'Parallel_Reduce("+", 0);
```

is short for:

```
Sum : constant Integer :=
    [ parallel for Value of A => Value ]'Reduce("+", 0);
```

2.6 Control over the degree of parallelism

Explicit chunk definition for parallel loops (AI12-0251-1) gives the user control of the degree of parallelism, for example if processing 100 elements of an array on a 20 core machine one may wish to have 10 logical threads of control (potentially executing on one core each) each processing a group of 10 elements. Such a group is referred to as a "chunk".

The optional chunk specification is placed, enclosed by round brackets, after the reserved word **parallel**.

In the more complicated form, an identifier is given that can be used within the parallel construct, for instance:

```
declare
  Partial_Sum : array (1 .. Max_CPUs_To_Use) of
                                  Integer := (others => 0);
begin
  parallel (Chunk in Partial_Sum'Range)
                                  for I in Arr'Range loop
    declare
      Z : Integer;
    begin
      Z := some_complicated_computation;
        ... -- Complex machinations
      Partial_Sum (Chunk) := @ + Arr (I)**2 * Z;
        ... -- Other stuff that also happens in this
        ... -- very complicated loop ...
    end;
  end loop;
  Sum := Partial_Sum'Reduce ("+", 0);
end;
```

This makes uses of a reduction expression, as described above, and @ is a shorthand for the left hand side (see the Others section).

In the simpler form, just the maximum number of logical threads of control to be created to execute the loop is given:

```
parallel (Max_CPUs_To_Use) for I in Arr'Range loop
  A (I) := B (I) + C (I);
end loop;
```

2.7 An example of the features working together

Not directly related to parallelism, *Index parameters in array aggregates (AI12-0061)* is also used in the example below as it is just such a useful new feature.

```
-- AI12-0241 Specifying Nonblocking for
--   Language-Defined Units
-- AI12-0079-3 Global-in and global-out annotations –
--   default Global => null" (i.e. no read or write of any --
--   global variable) for Pure packages
-- package Ada.Numerics.Generic_Elementary_Functions
--     with Pure, Nonblocking is
--     function Sqrt (X : Float_Type'Base) return
--                                   Float_Type'Base;
--   ...
with Ada.Numerics.Elementary_Functions;
-- …
declare
  Max_CPUs_To_Use : constant := 10;
  Max             : constant := 100;
  subtype Range_Type is Positive range 1 .. Max;
  type Float_Array_Type is
                       array (Range_Type) of Float;
  type Positive_Array_Type is
                       array (Range_Type) of Positive;
  -- AI12-0061 Index parameters in array aggregates
  -- modified by Container aggregates; generalized array
  -- aggregates (AI12-0212) to use [ ]
  Numbers : constant Positive_Array_Type :=
                              [ for I in Range_Type => I ];
  Squares         : Positive_Array_Type;
  Square_Roots    : Float_Array_Type;
  Sum_Of_Squares : Integer;
  -- AI12-0064-2 - Nonblocking subprograms
  -- AI12-0079-3 Global-in and global-out annotations
  function Square (P : Positive) return Positive
   with Nonblocking, Global => null;
  function Square (P : Positive) return Positive is
   (P**2);
begin
  -- Ignoring any risk of overflows…

  -- AI12-0119 Parallel operations
  -- Iteration over the elements of an array
  -- AI12-0251-1 Explicit chunk definition for parallel
  -- loops
  parallel (Max_CPUs_To_Use)
                              for I in Range_Type loop
    Squares      (I) := Square (I);
    Square_Roots (I) :=
     Ada.Numerics.Elementary_Functions.Sqrt (Float (I));
  end loop;
  -- AI12-0242 Shorthand Reduction Expressions for
  -- Objects (dependent on AI12-0262 Map-Reduce
  -- attribute)
  Sum_Of_Squares := Squares'Reduce ("+", 0);
end;
```

3 Contracts

Ada 83 introduced the generic contract model, whereby a contract is imposed on the types that can be used to instantiate a unit. Parameter modes, and subtypes with constraints, also dating from Ada 83, can be regarded as forms of contract. Ada 2012 added new forms:

preconditions, postconditions, type invariants and subtype predicates. Ada 202x adds further forms: aspect Nonblocking states that no potentially blocking operation should be called; global-in and global-out annotations to describe the use of global objects. Ada 202x also offers improvements to the existing forms of contract. One of the key benefits of contracts is that they allow checking by static analysis tools.

3.1 Defining Nonblocking

This form of contract is covered in the Parallelism section above since the driving reason for adding it was safe parallelism.

3.2 Defining access to global data

This is also covered in the Parallelism section above as again the driving reason for adding it was safe parallelism, but it is more generally useful, for dataflow analysis, for example, or simply documenting behaviour that some coding standards would ask for in a comment.

3.3 Contracts for container operations (AI12-0112)

The checks to be performed upon entry to a container's subprograms are now expressed using Ada 2012 preconditions rather than English.

Checking of the preconditions by the containers' users can be suppressed using:

```
pragma Suppress (Containers_Assertion_Check);
```

The aspects Nonblocking (*AI12-0064-2)* and Global *(AI12-0079-3)* are included in the contracts.

3.4 Pre- and Postcondition support

Partial aggregate notation (AI12-0127) introduces a new syntactic form of aggregate, the delta_aggregate. This allows one to update one or more fields of a composite object without having to specify every field. This will be particularly useful for postconditions, where one might want to check that only certain fields of a composite parameter had changed, for example:

```
procedure Twelfth (D : in out Date)
  with Post => D = (D'Old with delta Day => 12);
```

The values of the Year and Month components of the delta aggregate are the same as those of D'Old but the Day component is 12.

Stable properties of abstract data types (AI12-0187) adds the new aspect Stable_Properties to simplify the description of properties of an abstract data type (ADT), by making it easy to specify properties that are usually unchanged by most of the operations of the ADT. The classic example is the Mode of a file, which is unchanged by all of the operations other than Create/Open/Close/ Reset (and Set_Mode for streams). The stable properties are automatically included in the postconditions of all the primitive operations of the ADT, decreasing clutter and increasing the information that provers can use.

The aspect Stable_Properties can be given on a partial view or on a full type with no partial view, and also on primitive subprograms. The subprogram version can be used to override the type version when necessary for specific primitive subprograms of the type. The aspect is also applicable to class-wide versions.

The aspect is followed by a list of the stable property functions (comma-separated if more than one) for the primitive subprogram(s). *Syntax for Stable_Properties aspects (AI-0285)* tweaks the syntax so that the list is enclosed by round brackets, in the form of a positional aggregate, to avoid possible ambiguity in a list of aspects.

The postcondition(s) of the subprogram are modified with an item that verifies that the property is unchanged for each parameter of the appropriate type, unless that property is already referenced in the explicit postcondition (or inherited postcondition, in the case of class-wide postconditions).

To expand on the example of the Mode of a file, suppose that we wish many subprograms to behave as if they have a postcondition as in:

```
procedure Put (File : in File_Type; Str : in String)
  with Pre  => Mode(File) /= In_File,
       Post => Mode(File) = Mode(File)'Old;
```

Then rather than having to repeat this postcondition for numerous subprograms, if Ada.Text_IO could be rewritten in the form:

```
package Ada.Text_IO is
  type File_Type is private
    with Stable_Properties => (Is_Open, Mode);
…
```

then the declaration of Put could simply be:

```
  procedure Put (File : in File_Type; Item : in String)
  with Pre => Mode(File) /= In_File;
```

since we have stated that the Mode is a stable property. (Sadly we cannot change Ada.Text_IO – this is just an illustrative example for the future!)

Pre/Post for access-to-subprogram types (AI12-0220) allows Pre and Post aspects for access-to-subprogram types, so that contract information is available when calling a subprogram indirectly via an access value, as well as (or even instead of) when called directly. For example, to check that a parameter is even:

```
type T1 is access procedure (X : Integer)
  with Pre => X mod 2 = 0;

procedure Foo (X : Integer) is ... end;
…
  Ptr1 : T1 := Foo'Access;
begin
  Ptr1.all (222); -- Precondition check performed
```

Declare expressions (AI12-0236). As the power of expressions has grown, some felt that it would help to allow local constants and object renamings within an expression, to avoid repeated subexpressions. For example, the postcondition for Fgetc could be clarified from:

```
(if Stream.The_File (Stream.Cur_Position'Old) =
   EOF_Ch
 then Stream.Cur_Position = Stream.Cur_Position'Old
   and then Result = EOF
 elsif Stream.The_File (Stream.Cur_Position'Old) =
   ASCII.LF
 then Stream.Cur_Position = Stream.Cur_Position'Old
   and then Result = Character'Pos (ASCII.LF)
 else
   Stream.Cur_Position = Stream.Cur_Position'Old + 1
   and then Result = Character'Pos (Stream.The_File
     (Stream.Cur_Position'Old)))
```

to:

```
(declare
   Old_Pos : constant Position :=
                         Stream.Cur_Position'Old;
   The_Char : constant Character :=
                         Stream.The_File(Old_Pos);
   Pos_Unchg : constant Boolean :=
                         Stream.Cur_Position = Old_Pos;
 begin
   (if The_Char = EOF_Ch
     then Pos_Unchg and then Result = EOF
    elsif The_Char = ASCII.LF
     then Pos_Unchg and then Result =
                         Character'Pos(ASCII.LF)
    else
     Stream.Cur_Position = Old_Pos + 1
     and then Result = Character'Pos (The_Char)))
```

This uses the reserved words **declare** and **begin**, as for a block, but not **end**, as it is only used within parentheses.

Making 'Old more flexible (AI12-0280-2). Currently the following is illegal as the A.**all** in A.**all**'Old is "potentially unevaluated":

```
procedure Proc (A : access Integer)
   with Post =>
     (if A /= null then (A.all'Old > 1 and A.all > 1));
```

This is now relaxed. Thinking less of what it may not be possible to evaluate, but more of what CAN be evaluated in advance, the term "known on entry" is introduced to cover such expressions (the most obvious example being a static expression), and if it is possible to tell on entry to a subprogram that an X'Old need not be evaluated then it isn't. In the example, A is an **in** parameter of an elementary type (which includes access types) so it is passed by copy and cannot change, so if A is **null** on entry then A.**all**'Old would not be evaluated.

3.5 Aspects for Generic Formal Parameters

Previously, language-defined aspects were not allowed for generic formal parameters, but now several are allowed:

As mentioned in the Parallelism section, *Nonblocking subprograms (AI12-0064-2)* allows the Nonblocking aspect to be specified for generic formal parameters, and *Fixes for Nonblocking (AI12-0374-2)* clarifies that, at the point of instantiation, the Nonblocking aspects of the actual generic parameters are "and"-ed with the Nonblocking aspects of the operations within the generic.

The new aspect for types, Default_Initial_Condition (*Default_Initial_Condition for types AI12-0265* – see below) is allowed on generic formal private types.

Contracts for generic formal parameters (AI12-0272) allows Pre and Post on generic formal (nonabstract) subprograms. For example:

```
generic
   type Foo is ...
   with function Reduce (Obj : Foo) return Integer
     with Post => Reduce'Result in –9 .. 9;
package Gen is
   ...
end Gen;
```

Atomic, Volatile, and Independent generic formal types (AI12-0282). The aspects Atomic, Volatile, Independent, Atomic_Components, Volatile_Components, and Independent_Components can now be specified for generic formal types. The actual type must have a matching specification, though for backward compatibility reasons the actual types can be Atomic, etc., without the formal types necessarily matching.

3.6 Defaults for generic formal types (AI12-0205)

This provides easier and more natural generic instantiation. It uses the reserved words **or use**. For example:

```
generic
   type Item_Type is private;
   type Item_Count is range <> or use Natural;
                         -- New syntax using or use
   with function "=" (L, R : in Item_Type)
                                    return Boolean;
package Lists is
   ...
end Lists;
```

This allows the instantiator to be able to provide a type for the Item_Count, but it can simply be omitted in ordinary circumstances (in which case Natural would be used).

3.7 Default_Initial_Condition for types (AI12-0265)

A new contract aspect, Default_Initial_Condition, may be specified for a private type (or private extension). This is useful for checking that the default initialisation of an object has been performed as expected. After the successful default initialization of an object of the type, a default initial condition check is performed. In the case of a controlled type, the check is performed after the call to the type's Initialize procedure. For example:

```
package Sets is
   type Set is private
     with Default_Initial_Condition => Is_Empty (Set);
   function Is_Empty (S : Set) return Boolean;
   ...
end Sets;
```

3.8 Aspect No_Return for functions reprise (AI12-0269)

The aspect No_Return may now be specified for functions, not just procedures, but the reason that such a function never returns must be that it raises an exception (rather than containing an endless loop). As for procedures, there will be a check at compile time that the function does not contain any explicit return statements, and a check at run-time that it does not run into the final end.

4 Containers and Iterators

4.1 Stable Containers to reduce tampering checks (AI12-0111)

This attempts to address performance concerns about Ada containers, whilst maintaining their safety. Each container is given a nested package named Stable. This has similar contents to the parent package, but provides a variant of the container type that cannot grow or shrink, and omits operations that might tamper with elements of the container. Such a container can be created by calling the Copy function, or by creating a stabilised view of a normal container. The operations of the Stable package do not perform tampering checks as they are not needed, because the operations that tamper with elements have been omitted. The tampering checks are considered to incur the main performance overhead.

4.2 Contracts for container operations (AI12-0112)

As described in the Contracts section, the checks to be performed upon entry to a container's subprograms are now expressed using Ada 2012 preconditions rather than English. The aspects Nonblocking (*AI12-0064-2)* and Global *(AI12-0079-3)* are included in the contracts.

The new aspect Allows_Exit (see *Loop-body as anonymous procedure, AI12-0189* below) is applied to the container Iterate procedures.

All container operations now have versions with the container as the first parameter, thus allowing prefix notation to be used. The new versions include Element, Query_Element, Next and Previous for vectors and lists; Key, Element, Query_Element, Next and (if ordered) Previous for maps; Element, Query_Element, Next and (if ordered) Previous for sets; and Subtree_Node_Count, Element, Query_Element, Next_Sibling and Previous_ Sibling for multiway trees. If the cursor points to a different container to the one given, then Program_Error is raised.

Consider the following example:

```
package String_Indexes is
  new Indefinite_Hashed_Maps (Key_Type => String,
  …
  My_Index : constant String_Indexes.Map :=
                                  String_Indexes.Empty_Map;
function Long_Strings_Count
               (The_Index  : String_Indexes.Map;
                Min_Length : Positive) return Natural is
  Count : Natural := 0;
begin
  for My_Cursor in The_Index.Iterate loop
    if String_Indexes.Key (Position =>
                  My_Cursor)'Length >= Min_Length then
      Count := Count + 1;
    end if;
  end loop;
  return Count;
end Long_Strings_Count;
```

Note the incongruous mixing of OO-style prefix notation in The_Index.Iterate and traditional notation in String_Indexes.Key. In Ada 202x the latter can be replaced by The_Index.Key.

4.3 Use subtype_indication in generalized iterators (AI12-0156)

Ada 2012 added the ability to simplify:

```
Vec : Int_Vectors.Vector;
...
for I in Vec.Iterate loop
  Vec(I) := Vec(I) + 1;
end loop;
```

to:

```
Vec : Int_Vectors.Vector;
...
for E : T of Vec loop
   E := E + 1;
end loop;
```

where the optional : T acts as a comment to the reader that the subtype of element E is T (and the compiler verifies this comment). An optional subtype indication – though of the cursor not the element – can now also be given for the original **in** form of the loop, i.e.:

```
for I : Index in Vec.Iterate loop
  Vec(I) := Vec(I) + 1;
end loop;
```

where Index is the subtype of the loop parameter.

4.4 Indefinite Holders

Bounded_Indefinite_Holders (AI12-0254) adds a new container type, Bounded_Indefinite_Holder, which allows the storage of a (single) class-wide object without the use of dynamic memory allocation, for use in safety critical environments. Rather than having a bounded indefinite variant of every container, it is envisaged that this holder container would be used as a building block, e.g. in a container of such holder containers.

Compared with the existing Indefinite_Holder, there is an additional generic parameter:

```
   Max_Element_Size_in_Storage_Elements :
                                        Storage_Count;
```

If this is exceeded, Program_Error is raised.

Swap for Indefinite_Holders (AI12-0350) adds a Swap operation to both Indefinite_Holder and the new Bounded_Indefinite_Holder. For the former this avoids the overhead of copying the element (and any associated Adjust/Finalize).

```
   procedure Swap (Left, Right : in out Holder)
     …
```

4.5 Loop-body as anonymous procedure (AI12-0189)

A loop body can be used to specify the implementation of a procedure to be passed as the actual for an access-to-subprogram parameter, when used in the context of a special kind of for-loop statement, whose iterator_ specification is given by a procedure_iterator.

This can be used for iterating over Directories and Environment variables, or iterating through a map-like container over the keys. Dedicated mechanisms were proposed for these (*AI12-0009* and *AI12-0188*, respectively), but it was considered more useful to add a more general mechanism. For example:

```
   for (Name, Val) of Ada.Environment_Variables.Iterate
                                              (<>) loop
   -- "(<>)" is optional because it is the last parameter
     Put_Line (Name & " => " & Val);
   end loop;

   for (C : Cursor) of My_Map.Iterate loop
     Put_Line (My_Key_Type'Image (Key (C)) & " => " &
                   My_Element_Type'Image (Element (C)));
   end loop;
```

An exit, return, goto, or other transfer of control out of the loop is only allowed if the named procedure has new aspect Allows_Exit with value True. Even if Allows_Exit is False, the loop can still end prematurely due to the propagation of an exception.

Bounded errors associated with procedural iterators (AI12-0326-2) extends this to make it a bounded error for an Allows_Exit subprogram to call the loop body procedure from an abort-deferred operation (unless the whole loop_statement was within this same abort-deferred operation), as this would interfere with implementing a transfer of control.

It is also adds the reserved word **parallel** to the syntax for procedural iterators, and makes it a bounded error to call a loop-body procedure from multiple logical threads of control unless **parallel** is specified.

4.6 Container aggregates

Currently, it is quite tedious to initialise a container, one has to create it as an empty container and then add elements one at a time, as in:

```
   X : My_Set := Empty_Set;
   Include (X, 1);
   Include (X, 2);
   Include (X, 3);
```

Container aggregates; generalized array aggregates (AI12-0212) adds positional container aggregates. These allow the above to be replaced by simply:

```
   X : My_Set := [ 1, 2, 3 ];
```

Note that this uses square brackets not round brackets (parentheses). This allows the use of [ ] to indicate an empty container, analogous to "" indicating an empty string.

This is achieved using the new aspect Aggregate to indicate the appropriate function for returning an empty container of the particular container type, and also the appropriate procedure for adding an element to the particular container type.

For example:

```
   type Set is tagged private
     with -- Ada 2012 has these
        Constant_Indexing => Constant_Reference,
        Default_Iterator    => Iterate,
        Iterator_Element   => Element_Type,
        …  -- but this is new
        Aggregate           => (Empty            => Empty,
                                Add_Unnamed => Include),
        …
```

Originally an Empty_<Container> constant was asked for in *AI12-0212*, but *Empty function for Container aggregates (AI12-0339)* tweaked this such that now an Empty function is given instead, so as to allow a Capacity parameter for those container types that have the notation of capacity (e.g. Vectors).

Add_Unnamed requires a two parameter procedure for adding a single element. As the existing Append procedures of vectors and lists required a third, Count, parameter, *Contracts for container operations (AI12-0112)* added a procedure without the Count parameter (at the time called Append_One) to the Vectors container, and *List containers need Append_One (AI12-0391)* did the same for the Doubly_Link_Lists container. *Ambiguities associated with Vector Append and container aggregates (AI12-0400)* then decided it was cleaner for the new procedures to be overloadings of Append, and to remove the default for Count (of := 1) from the original Append procedures.

Iteration is also possible within the container aggregate, for example to create a set whose elements all have double the value of the corresponding elements of another set:

```
   Doubles_Set : My_Set := [ for Item of X => Item * 2 ];
```

Note that this uses similar syntax to that introduced by *Index parameters in array aggregates (AI12-0061)* (see the Others section).

And – prepare yourself for a shock! – array aggregates are also allowed to use square brackets as an alternative to round brackets (parentheses). This is to emphasise the similarity in characteristics between containers and arrays, allow the use of [ ] for an empty array, and allow the use of positional notation for a single element array. Remember that

```
Two_Array : array (1 .. 2) of Positive := (1, 2);
```

is allowed, but not :

```
One_Array : array (1 .. 1) of Positive := (1);
```

Unfortunately introducing aggregates for containers introduced an ambiguity – for the Append, Insert and Prepend operations of vectors and lists, if the element type is a record then it is unclear whether it is an element or another vector or list that is being added. *Ambiguities associated with Vector Append and container aggregates (AI12-0400)* renames the operations for adding a list to Append_List, Insert_List and Prepend_List, and similarly for vectors. Note that this is a backward incompatibility.

4.7 Iterator filters (AI12-0250)

When iterating through a container, it is often required to filter the results to only return those values that meet some condition. This feature makes use of the keyword **when**, for example:

```
S : constant Set :=
                        (for E of C when E mod 2 = 1 => E);
```

to obtain all the odd elements of Container C.

4.8 Parallel Container Iterator filters (AI12-0266)

This is covered in the Parallelism section above.

5 Internationalisation

5.1 Additional internalization of Ada (AI12-0021)

This adds child packages Wide_File_Names and Wide_Wide_File_Names for each I/O package (i.e. Sequential_IO, Direct_IO, Text_IO and Stream_IO), containing just those operations that take a filename as a parameter, and Wide_ and Wide_Wide_ versions of Ada.Directories, Ada.Command_Line and also of Ada.Environment_Variables.

6 Real-Time

A number of AIs were discussed by, or arose from, the 19th International Real-Time Ada Workshop, as reported on in the Vol. 39, No. 2, March 2018 edition of the AUJ:

- *Thread-safe Ada libraries (AI12-0139)*. This was subsequently dropped;
- *Deadline Floor Protocol (AI12-0230)*;
- *Compare-and-swap for atomic objects (AI12-0234)*;
- *Admission Policy Defined for Acquiring a Protected Object Resource (AI12-0276)*;
- *Dispatching Needs More Dispatching Points (AI12-0279)*;
- *CPU Affinity for Protected Objects (AI12-0281)*;
- *Atomic and Volatile generic formal types (AI12-0282)*.

IRTAW also proposed an extended version of the Ravenscar profile called Jorvik.

6.1 Defining Nonblocking

This is covered in the Parallelism section above as the driving reason for adding it was safe parallelism, but it is more generally useful, for timing analysis or deadlock avoidance, for example, or simply documenting behaviour that some coding standards would ask for in a comment.

6.2 Exact size access to parts of composite atomic objects (AI12-0128)

Memory accesses to subcomponents of an atomic composite object must read or write the entire object. For example:

```
type Status is
  record
    Ready : Boolean;
    Length : Integer range 0 .. 15;
  end record;
for Status use
  record
    Ready at 0 range 0 .. 0;
    Length at 0 range 1 .. 5;
  end record;
Status_Register : Status
  with Address => ...,
       Size => 32,
       Atomic => True;
if Status_Register.Ready then -- Reads entire register
  null;
end if;
Status_Register.Length := 10; -- Prereads entire register,
                              -- then writes entire register.
```

This is useful for controlling accesses to memory mapped device registers, which often require reads or writes to be to the entire register.

6.3 Max_Entry_Queue_Length aspect for entries (AI12-0164)

The new aspect Max_Entry_Queue_Length for an entry declaration specifies the maximum number of callers allowed on that entry. This facilitates timing analysis and should be useful for new restricted tasking profiles besides Ravenscar.

Violation of this restriction results in the raising of Program_Error at the point of the call or requeue.

The value specified for the Max_Entry_Queue_Length aspect for an entry must be no higher than any specified for the corresponding type, and both must be no higher than the Max_Entry_Queue_Length partition-wide restriction. These are checked at compilation.

6.4 Deadline Floor Protocol (AI12-0230)

This updates the EDF (Earliest Deadline First) policy in line with the latest thinking from the IRTAW Workshops. It now uses the Deadline Floor Protocol (DFP) in preference to the Stack Resource Protocol (SRP). This should not result in any backward compatibility problems as it is believed that currently no Ada implementations support EDF.

6.5 Atomic Operations

A family of atomic operations is added to optional Annex C Systems Programming by *Compare-and-swap for atomic objects (AI12-0234)*, *Support for Arithmetic Atomic Operations and Test and Set (AI12-0321)* and *Add a modular atomic arithmetic package (AI12-0364)*.

In systems where processors communicate via shared memory, there are two common methods of synchronisation. Thus "Compare-and-swap" atomically compares the contents of a memory location with a given value and, only if they are the same, modifies the contents of that memory location to a given new value. "Test-and-set" modifies the contents of a memory location and returns its old value in a single atomic operation. These can then be used to construct spin locks. Such instructions are often available in the hardware. This AI makes them available in a more portable manner, assuming of course that the underlying hardware provides them. Note that which instructions can be performed atomically can vary even between processors in the same family.

These AIs add a number of library packages, namely

```
System.Atomic_Operations.Test_And_Set,
```

and the generic packages

```
System.Atomic_Operations.Exchange,
System.Atomic_Operations.Integer_Arithmetic,
System.Atomic_Operations.Modular_Arithmetic
```

The last three packages are generic so that they can be instantiated with an actual of the appropriate size for the memory architecture in use.

Compare-and-swap:

```
generic
   type Atomic_Type is private with Atomic;

package System.Atomic_Operations.Exchange
     with Pure, Nonblocking is
   function Atomic_Exchange
               (Item  : aliased in out Atomic_Type;
                Value : Atomic_Type) return Atomic_Type
     with Convention => Intrinsic;
   function Atomic_Compare_And_Exchange
               (Item  : aliased in out Atomic_Type;
                Prior : aliased in out Atomic_Type;
                Desired : Atomic_Type) return Boolean
     with Convention => Intrinsic;
   function Is_Lock_Free (Item : aliased Atomic_Type)
                                        return Boolean
     with Convention => Intrinsic;
end System.Atomic_Operations.Exchange;
```

Atomic_Exchange atomically assigns the value of Value to Item, and returns the previous value of Item.

Atomic_Compare_And_Exchange first evaluates the value of Prior. It then performs the following steps as part of a single indivisible operation:

- evaluates the value of Item;
- compares the value of Item with the value of Prior;
- if equal, assigns Item the value of Desired;
- otherwise, makes no change to the value of Item.

After these steps, if the value of Item and Prior did not match, Prior is assigned the original value of Item, and the function returns False. Otherwise, Prior is unaffected and the function returns True.

Is_Lock_Free returns whether all the operations of the child package can be provided lock-free for a given object.

An example of a spin lock using Atomic_Exchange:

```
   type Atomic_Boolean is new Boolean with Atomic;
   package Exchange is
     new Atomic_Operations.Exchange
               (Atomic_Type => Atomic_Boolean);
   Lock : aliased Atomic_Boolean := False;
...

begin -- Some critical section, trying to get the lock:
      -- Acquire the lock
   while Exchange.Atomic_Exchange
               (Item => Lock, Value => True) loop
     null;
   end loop;
   ...                  -- Do stuff
   Lock := False;       -- Release the lock
end;
```

For non-preemptive scheduling, it might be appropriate to call Ada.Dispatching.Yield rather than having a **null** statement.

Test-and-set:

```
package System.Atomic_Operations.Test_And_Set
   with Pure, Nonblocking is
   type Test_And_Set_Flag is
              mod<implementation-defined>
     with Atomic, Default_Value => 0,
          Size => <Implementation-Defined>;
   function Atomic_Test_And_Set
            (Item : aliased in out Test_And_Set_Flag)
                                         return Boolean
     with Convention => Intrinsic;
   procedure Atomic_Clear
            (Item : aliased in out Test_And_Set_Flag)
     with Convention => Intrinsic;
   function Is_Lock_Free
     (Item : aliased Test_And_Set_Flag) return Boolean
     with Convention => Intrinsic;
end System.Atomic_Operations.Test_And_Set;
```

Atomic_Test_And_Set performs an atomic test-and-set operation on Item. Item is set to some implementation-defined non-zero value. The function returns True if the previous contents were non-zero, and otherwise returns False.

Atomic_Clear performs an atomic clear operation on Item. After the operation, Item contains zero.

An example of a spin lock using Atomic_Test_And_Set:

```
begin -- Some critical section, trying to get the lock:
   -- Acquire the lock
   while Atomic_Test_And_Set (Item => Lock) loop
      null;
   end loop;
   ... -- Do stuff
  Atomic_Clear (Item => Lock); -- Release the lock
end;
```

Atomic arithmetic:

```
generic
   type Atomic_Type is range <> with Atomic;
package System.Atomic_Operations.Integer_Arithmetic
      with Pure, Nonblocking is
   procedure Atomic_Add
           (Item   : aliased in out Atomic_Type;
            Value : Atomic_Type)
     with Convention => Intrinsic;
   procedure Atomic_Subtract
          (Item   : aliased in out Atomic_Type;
            Value : Atomic_Type)
     with Convention => Intrinsic;
   function Atomic_Fetch_And_Add
           (Item   : aliased in out Atomic_Type;
            Value : Atomic_Type) return Atomic_Type
     with Convention => Intrinsic;
   function Atomic_Fetch_And_Subtract
           (Item   : aliased in out Atomic_Type;
            Value : Atomic_Type) return Atomic_Type
     with Convention => Intrinsic;
   function Is_Lock_Free (Item : aliased Atomic_Type)
                                          return Boolean
    with Convention => Intrinsic;
end System.Atomic_Operations.Arithmetic;
```

As one might expect, Atomic_Add and Atomic_Subtract atomically perform add and subtract, whereas Atomic_Fetch_And_Add and Atomic_Fetch_And_Subtract additionally return the original value of the Item.

Modular arithmetic:

in this case, the package

```
System.Atomic_Operations.Modular_Arithmetic
```

has the same declaration as

```
System.Atomic_Operations.Integer_Arithmetic,
```

except that the formal parameter is:

```
type Atomic_Type is mod <> with Atomic;
```

6.6 Admission policy defined for acquiring a protected object resource (AI12-0276)

On multiprocessor systems, a spin lock is typically used to gain access to a protected object in order to execute a protected action. Most multiprocessor locking algorithms prescribe that if there is more than one request competing for the same resource, the requests are served in FIFO order. This bounds the time it takes for a lower priority task to gain access to a protected object. With Ravenscar all tasks are statically allocated to processors, and if each protected object used by tasks on different processors is given a high ceiling priority then the blocking time for each task can be computed.

Previously the language did not presume any ordering or queuing for tasks competing to start a protected action, this AI adds:

```
pragma Admission_Policy (FIFO_Spinning);
```

to specify that FIFO_Spinning is used. A task will inherit the ceiling priority of the protected object. Other, implementation-defined, Admission_Policies may also be specified.

6.7 Nonpreemptive dispatching needs more dispatching points (AI12-0279)

In non-preemptive dispatching there needs to be sufficient points where rescheduling can occur, so as to restrict the amount of time that a low priority task can block a higher priority task. If the low priority task "gets lucky" and the entries it calls are open, and suspension objects true, it can be some time before a rescheduling point occurs.

The solution to this is to define a Yield aspect that can be specified for a subprogram. If a reschedule has not occurred within a call of the subprogram, then one is inserted at the return from the subprogram.

6.8 CPU Affinity for Protected Objects (AI12-0281)

This allows the CPU aspect to be applied to a protected type, not just a task type. If all tasks that invoke protected operations of a protected object are on the same CPU as the protected object then it is possible for the runtime to avoid the overhead of acquiring a lock, and also avoid the risk of deadlock.

A Program_Error is raised if a task on one CPU attempts to invoke a protected operation of a protected object on another CPU.

6.9 Atomic, Volatile, and Independent generic formal types (AI12-0282)

This is covered in the Contracts section above.

6.10 Jorvik Profile (AI12-0291)

The new Jorvik profile is less restrictive than the Ravenscar profile, but still allows timing and storage analyses. Most of the restrictions are the same, but restrictions

```
No_Implicit_Heap_Allocations,
No_Relative_Delay,
Max_Protected_Entries => 1,
No_Dependence => Ada.Calendar
No_Dependence => Ada.Synchronous_Barriers
```

are omitted and the restriction Simple_Barriers is replaced by the weaker Pure_Barriers.

Restriction Pure_Barriers (AI12-0290) defines Pure_ Barriers. Such barriers do not have to be simple Boolean local variables, but can be more complex Boolean expressions, as long as they do not have side effects, exceptions, or recursion. Additionally, the ’Count attribute is allowed in entry barriers, not just protected entry barriers.

6.11 Fixes for Atomic and Volatile (AI12-0363)

’Access may not be taken of a non-atomic subcomponent of an atomic object.

The nesting of atomic objects gives much scope for confusion, but to disallow them would be backwardly incompatible, so a new aspect Full_Access_Only is added. This can be applied to atomic and volatile types and objects to indicate that no atomic (or full access) objects are permitted as subcomponents. If any subcomponent of a full access object is accessed, then the whole object has to be accessed, by an atomic read followed by an atomic write.

An Atomic aspect of True now additionally indicates that Volatile and Independent are True.

7 Others

The heading "Others" is not meant to imply that the features in this section are less important; indeed this section includes some of the most useful new features of Ada 202x. They are likely to be amongst the first to be implemented, and the first that programmers will want to use.

Some of the changes are to tidy up inconsistencies in the language, such as *Missing operations of static string types (AI12-0201)* and *Make objects more consistent (AI12-0226)*.

7.1 'Image for all types (AI12-0020)

It must be a shock to programmers coming from other languages, such as Python, that in Ada one couldn't directly output the value of a composite, but had to laboriously write one's own routine to do it field by field, then repeat if there was nesting of composites. Such a mechanistic process is more efficiently performed by a compiler than a programmer. And remember that, prior to *Add Object’Image (AI12-0124)*, included in the Ada 2012 Technical Corrigendum, there was also the tedium of having to look up the subtype for an object and use My_Subtype’Image (My_Object) to obtain the image of an object.

This AI adds the attribute 'Image for all types. It should be a boon for debugging.

Following on from this, *Image attributes of language-defined types (AI12-0304)* requires that 'Image works for the language defined container types. This uses the new [ ] array aggregate syntax from *Container aggregates; generalized array aggregates (AI12-0212)*. For Maps it uses the form of a named array aggregate, e.g.:

```
[ Key1 => Value1, Key2 => Value2 ]
```

for Trees the form is a positional array aggregate, e.g.:

```
[ [ 1, 2 ] , [ 111, 222, 333 ] ]
```

for null containers the form is a null array aggregate, i.e.:

```
[ ]
```

7.2 The Fortran Annex needs updating to support Fortran 2008 (AI12-0058)

The Fortran section of mandatory Annex B has been updated to support Fortran 2008, in particular better support for double precision complex arithmetic. Permissions corresponding to non-standard extensions, or implementation advice that is now considered to be bad practice, have been removed.

7.3 Object_Size attribute (AI12-0059)

Users have been after this since 1983! S'Object_Size denotes the size of an object of subtype S. It can be specified, but must be specified to a value that the compiler is able to allocate (usually an entire storage unit for most implementations).

S'Object_Size is an improvement on S'Size (which cannot be redefined without breaking existing code). Reading 'Size is not terribly useful as it just gives the theoretical minimum number of bits required for a value of a given range, not the number of bits that the compiler is actually going to allocate to an object of the type. Specifying S'Size just gives a minimum, the compiler may allocate more.

7.4 Index parameters in array aggregates (AI12-0061)

Consider:

```
subtype Index is Positive range 1 .. 10;
type Array_Type is array (Index) of Positive;
Squares_Array : Array_Type := (for I in Index => I * I);
```

This provides a means of creating an aggregate when the element type is limited, provides a better means of initialising an array with a type invariant, and should be useful for everyday programming.

7.5 Static expression functions (AI12-0075)

The aspect Static is introduced. It can only be applied to an expression function, and requests that it be regarded as a static function.

If called in a context that requires the expression function to be static, such as in a static expression, then its actual parameters need to be static. For example, if we declare:

```
function If_Then_Else (Flag : Boolean;
                        X, Y : Integer) return Integer is
  (if Flag then X else Y) with Static;
```

and then attempt to declare:

```
X : constant := If_Then_Else (True, 37, 1 / 0); -- Error.
```

we get an error at compile time since 1/0 is not a static expression.

7.6 Aggregates and variant parts (AI12-0086)

A discriminant that controls a variant can now be non-static if the subtype of the discriminant is static and all values belonging to that subtype select the same variant. For example:

```
type Enum is (Aa, Bb, Cc, ..., Zz);
subtype S is Enum range Dd .. Hh;
type Rec (D : Enum) is record
  case D is
    when S => Foo,
      Bar : Integer;
    when others =>
      null;
  end case;
end record;
function Make (D : S) return Rec is
begin
  return (D => D, Foo => 123, Bar => 456);
end;
```

7.7 Add @ as an abbreviation for the LHS of an assignment (AI12-0125-3)

This new feature, which proved to be rather controversial with those who are used to Ada being verbose, uses a single character placeholder for the left hand side of an assignment.

```
My_Package.My_Array(I).Field :=
  My_Package.My_Array(I).Field + 1;
```

could be shortened to:

```
My_Package.My_Array(I).Field := @ + 1;
```

The above is similar in function to the += of the C family of languages. The Ada feature is more powerful though, being able to handle expressions such as series expansions. Here are a couple of examples:

```
My_Package.My_Array(I).Field :=
  My_Package.My_Array(I).Field ** 3 +
  My_Package.My_Array(I).Field ** 2 +
  My_Package.My_Array(I).Field;
```

could be shortened to:

```
My_Package.My_Array(I).Field := @ ** 3 + @ ** 2 + @;
```

and:

```
My_Package.My_Array(I).Field :=
  Natural'Min (My_Package.My_Array(I).Field, 1000);
```

could be shortened to:

```
My_Package.My_Array(I).Field := Natural'Min (@, 1000);
```

7.8 Aggregates of Unchecked_Unions using named notation (AI12-0174)

Given that it is generally regarded as good practice to use named notation rather than positional notation, it was somewhat bizarre that Unchecked_Unions only allowed the latter. Both are now allowed. For example:

```
type Data_Kind is (C_int, C_char);
type C_Variant (Format : Data_Kind := C_int) is record
    case Format is
      when C_int =>
        int_Val : C.int;
      when C_char =>
        char_Val : C.char;
    end case;
end record
  with Unchecked_Union, Convention => C;
Int1 : C_Variant := (C_int, 12); -- Always OK
Int2 : C_Variant := (Format => C_int, int_Val => 12);
            -- Was illegal, now OK
```

7.9 Preelaborable packages with address clauses (AI12-0175)

Packages with aspect Preelaborate can now contain certain simple functions known to the compiler, i.e. an instance of Unchecked_Conversion, a function declared in System.Storage_Element, or the functions To_Pointer and To_Address declared in an instance of System.Address_to_Access_Conversions. This allows the declaring of objects with an address clause within a preelaborable package, which can be very useful for small embedded systems.

7.10 Missing operations of static string types (AI12-0201)

Relational operators and type conversions of static string types are now static.

Static membership tests for strings, e.g. S **in** "abc", were already allowed; static equality tests for strings, e.g. S = "abc", are now also allowed.

7.11 Big Numbers

Predefined Big numbers support (AI12-0208) defines a package Big_Numbers and various child packages to support arbitrary precision arithmetic.

Changes to Big_Integer and Big_Real (AI12-0366) updates this in the light of implementation experience.

In order to make Big Numbers easier to use, *User-defined numeric literals (AI12-0249)* allows the user to define numeric literals to be used with a (non-numeric) type, using aspects Integer_Literal and Real_Literal to identify a function that will do the interpretation. For example:

```
type Big_Integer is private
  with Integer_Literal => Big_Integer_Value;
function Big_Integer_Value (S : String)
                                   return Big_Integer;
...
Y : Big_Integer := -3;
```

This is equivalent to:

```
Y : Big_Integer := - Big_Integer_Value ("3");
```

Named Numbers and User-Defined Numeric Literals (AI12-0394) extends this to allow named numbers to be used with user-defined literals.

User-defined string literals (AI12-0295) allows the user to define string literals to be used with a non-string type, using aspect String_Literal to identify a function that will do the interpretation. For example:

```
type Varying_String is private
  with String_Literal => To_Varying_String;
function To_Varying_String (Source :
                          Wide_Wide_String)
  return Varying_String;
...
X : constant Varying_String := "This is a test";
```

This is equivalent to:

```
X : constant Varying_String :=
     To_Varying_String
      (Wide_Wide_String'("This is a test"));
```

7.12 Objects, Values, Conversions and Renaming

There are several AIs that aim to make Ada more consistent in its treatment of type conversions and qualified expressions, objects and values.

A qualified expression makes a predicate check (AI12-0100).

In Ada 2012, both a type conversion and a qualified expression perform a range check (if there is a constraint), but only the former performs a predicate check (if enabled). In Ada 202x both perform both checks. For example:

```
subtype Even is Natural
  with Dynamic_Predicate => Even mod 2 = 0;
Var   : Natural;
Three : constant Natural := 3;
Var := Even(Three); -- Predicate check, thus raises
                    -- Assertion_Error
Var := Even'(Three); -- Previously no predicate check,
                     -- now is.
```

Make objects more consistent (AI12-0226).

In Ada 2012 a type conversion between two tagged types (termed a view conversion) gives an object, which can be renamed, whereas a type conversion between two untagged types (termed a value conversion) gives a value, which cannot. In Ada 202x a value conversion of an object is added to the list of things deemed to be an object, making it consistent with a qualified expression. If we have

```
Max : constant Natural := 10;
```

then the following are now all legal:

```
Ren1 : Natural renames Max;
-- Legal

Ren2 : Natural renames Natural'(Max);
-- Qualified expression, legal

Ren3 : Natural renames Natural(Max);
-- Value conversion, was illegal, now legal
```

Renaming values (AI12-0383) takes this further by allowing values to be renamed anyway, besides objects.

Make subtype_mark optional in object renames (AI12-0275).

This makes the subtype optional in object renaming declarations. The expression will be correctly typed as long as the right hand object can be resolved to only one specific type.

It has long been an irritant that writing a renaming declaration required looking up the subtype of the object, and giving the subtype can be misleading anyway. The reader may be surprised to find that the following is valid Ada:

```
  subtype My_Subtype       is Integer range 3 .. 5;
  subtype Garbage_Subtype is Integer range -19 .. -7;
  X : My_Subtype;
  Y : Garbage_Subtype renames X;
begin
  case Y is
    when 3 .. 5 =>
      pragma Assert (Y in 3 .. 5);
  end case;
```

The subtype given in the renaming merely has to be a subtype of the same type as the object being renamed. The constraints, null exclusions, and predicates of the subtype given in the renaming are ignored; instead they are taken from the subtype of the object being renamed. Note that the case statement has to have alternatives for each possible value of the subtype of X, not of the supposed subtype of Y.

Requiring the subtypes to match has been considered in the past, but obscure problems have always prevented this. For backward compatibility reasons the subtype may still be given, but it is (usually) no longer required.

Renaming of a qualified expression of a variable (AI12-0401).

Ada 2012 allows a qualified expression to be renamed (though possibly this had not been implemented by anyone until recently). In the following, three subtypes are involved:

```
subtype Subtype_1 is Integer range …;
subtype Subtype_2 is Integer range …;
subtype Subtype_3 is Integer range …;
X : Subtype_1;
Y : Subtype_2 renames Subtype_3’(X);
```

As described under *Make subtype_mark optional in object renames (AI12-0275)*, Subtype_2 is largely irrelevant. Typically, a qualified expression is used to confirm which type is meant when there is a possible ambiguity, as in:

```
type Traffic_Light is (Red, Amber, Green);
type Gemstones is (Amber, Ruby, Topaz);
… Traffic_Lights'(Amber) …
… Gemstones'(Amber) …
```

But in the above it is not a case of X being renamed, and Subtype_3' just being there to confirm the type; instead it is the whole qualified expression Subtype_3'(X) that is the object being renamed. Thus Y takes its constraints, null exclusions, and predicates from Subtype_3. This could be problematic if Subtype_3 has a narrower range than Subtype_1, as in the following:

```
  X : Natural := …;
  Y : Positive renames Positive'(X);
begin
  X := 0; -- Alert!!
  case Y is
    when 1 .. Positive'Last =>
      pragma Assert (Y > 0);
  end case;
```

X is set to a value outside of the range of Y, effectively bypassing the range check at the time of renaming, which is meant to be a one-off affair, and not have to be repeated at each subsequent use of Y.

To fix this hole, *Renaming of a qualified expression of a variable (AI12-0401)* requires Subtype_3 to match either Subtype_1 or its base type (in this case, Integer). Note that the converse problem of writing a value to Y outside of the range of X does not arise as a qualified expression gives a constant (i.e. read-only) view.

7.13 Attributes for fixed point types (AI12-0362)

An implementation is permitted to support Floor, Ceiling, and rounding attributes for fixed point types. The AI came in rather late in the day to mandate support, but there is sufficient support to add something.

8 Conclusion

Ada 202x will bring Ada into a new decade. May many more projects be delivered successfully using it!

Tools to get you there. Safely.
Ada and The GNAT Pro High-Integrity Family
www.adacore.com
AdaCore
The GNAT Pro Company

Real-time Issues in the Ada Parallel Model with OpenMP

Luis Miguel Pinho

Polytechnic Institute of Porto, Portugal; lmp@isep.ipp.pt

Sara Royuela, Eduardo Quiñones

Barcelona Supercomputing Center, Spain; {sara.royuela,eduardo.quinones}@bsc.es

Abstract

The current proposal for the next revision of the Ada language considers the possibility to map the language parallel features to an underlying OpenMP runtime. As previously presented, and discussed in previous workshops, the works on fine-grain parallelism in Ada map well to the OpenMP tasking model for parallelism. Nevertheless, and although the general model of integration, and the semantic constructs are already reflected in the proposed revision of the standard, the integration of these new features with the Real-Time Systems Annex of Ada is still not complete. This paper presents an overview of what is supported and the still open issues.

1 Introduction

The existent proposal to extend Ada with a fine-grained parallelism model is based on the notion of logical threads of control: A single task, when within the context of a parallel construct, can represent multiple logical threads of control which can proceed in parallel (LRM 202X, ch. 9) [1]. In this revision of the language, parallel constructs can be found as parallel loops (LRM 5.5) and parallel bocks (LRM 5.6.1), as well as reduction expressions (LRM 4.5.10) and iterators (LRM 5.5.2) [1]. This structured approach to parallelism implements a fully-strict fork-join model, as defined in the tasklet model [2].

In parallel to the development of the Ada language specification, works have been made that demonstrate the suitability of the OpenMP tasking model 1 to support the parallel features of Ada [3], a topic which has been discussed in the last IRTAW workshop [4]. Recently this proposal has gained attraction, and a prototype implementation is validating the possibility [5].

Nevertheless, although the current draft of the Ada 202X standard already includes the parallelism support, and there exists validation with a prototype implementation, the use of the parallel features together with the Real-Time Systems Annex of Ada is still unclear. In order to allow that the parallel features are used in a real-time application, care must be taken that (i) the language model allows for a correct execution according to the LRM Annex D specifications, and (ii) the OpenMP and Ada runtimes are correctly integrated.

[1] The term task in OpenMP is not related to Ada tasks, as OpenMP tasks are lightweight parts of the code that can be executed in parallel by worker threads.

It is already too late to propose changes to the Real-Time Systems Annex to accommodate the use of the fine-grain parallel features, therefore this paper starts a process to analyse how this can be later performed (e.g., via a technical report). It is nevertheless important to determine if it is not necessary to update the annex taking into consideration that a task can now represent multiple logical threads of control. Note that in many places of the standard, the word *task* is replaced by the *logical threads of control*, but in Annex D only a note is made (LRM D2.1 4/5) [1].

On the other hand, the growing demand for high-performance computing in embedded systems (e.g., autonomous driving) is pushing the use of high-level parallel programming models, such as OpenMP, to exploit embedded hardware. For that reason, there is a significant effort in the OpenMP community to extend the use of OpenMP to uses other than High-Performance Computing, such as MPSoCs [6,7]. In this line, a discussion forum within the OpenMP Architecture Review Board has been made available to tackle the topics regarding real-time systems, considering timing constraints and functional safety.

Overall, the plan is to continue working in the real-time interactions between Ada and OpenMP, in parallel to the real-time OpenMP work, with the objective of proposing a technical report to be included in Ada 203X (or before). Note, nevertheless, that the open issues, and potential solutions, are also applicable to non-OpenMP implementations of the Ada parallel model.

2 System Model

The integration of real-time and parallelism in Ada is a complex task, due to the complexity of the Ada concurrency model, and the richness of its real-time features. Therefore, in the current work we are assuming a simplified Ada runtime, as well as the restriction to use only the OpenMP tasking model (no use of OpenMP threading constructs).

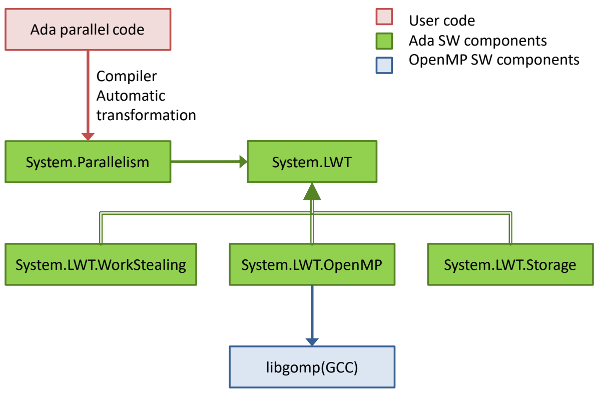


Figure 1. Ada parallel model implementation by AdaCore.

Therefore, the work considers architectures based on both homogenous and heterogenous processors, restricted to:

1. Homogenous processors, with a multicore real-time Operating System (OS), supporting global scheduling, with a single ready queue across processors.
2. Heterogenous processors, with homogenous host processors (using an OS as in 1) and a heterogenous fabric, which is either opaque to OpenMP, and is treated as an independent accelerator (using message passing), or also managed by an OpenMP runtime, with a minimal or eventually with no OS.

The plan is to later remove these restrictions, supporting other models, in particular with partitioned scheduling (one fixed-priority scheduler per core, with no thread migration), and EDF and server-based scheduling in the host part of heterogenous processors.

3 Current Status

The current work on implementing the proposed parallel model for Ada includes the implementation on top of a generalized lightweight parallel model, which can be mapped to OpenMP (or other implementations). Figure 1 shows the high-level architecture of the approach used by AdaCore in their prototype implementation [5]. There, the Ada 202X parallel model is automatically transformed by the compiler into generic calls of an intermediate API implemented in System.Parallelism. This API uses the System.LWT module to initialize and finalize parallel regions, as well as to implement the specific light-weight thread support, currently for (a) OpenMP, (b) a work-stealing approach, and (c) a storage approach (for reducing the heap usage).

In this approach, Ada tasks with no parallelism are mapped to just one thread, and each Ada task enclosing a parallel region is associated with a thread pool provided by OpenMP to execute the fine-grain parallel constructs (Figure 2). This implementation entails a limitation on the overall scheduling of the application, and this is related to how OpenMP defines parallel regions. In this regard, the team of threads that is associated to each parallel region in OpenMP, constitutes a black box for the rest of teams in OpenMP [8], as well as for the Ada scheduler. This may cause the execution to be non-conforming with specific real-time requirements, such as keeping a work-conserving strategy, or ensuring a priority-driven scheduler. Different solutions have been proposed to tackle this issue, some involving alterations to the OpenMP specification to avoid the black-box nature of OpenMP teams [9], another enforcing specific implementations of the OpenMP specification regarding the mapping to the underlying resources [10], and finally, and most relevant to this work, another offering a specific templated execution that forces a unique OpenMP team of threads to be accessed from any Ada task [11]. The prototype implementation propagates Ada priorities to the threads created for the OpenMP parallel, which allows to solve this problem.

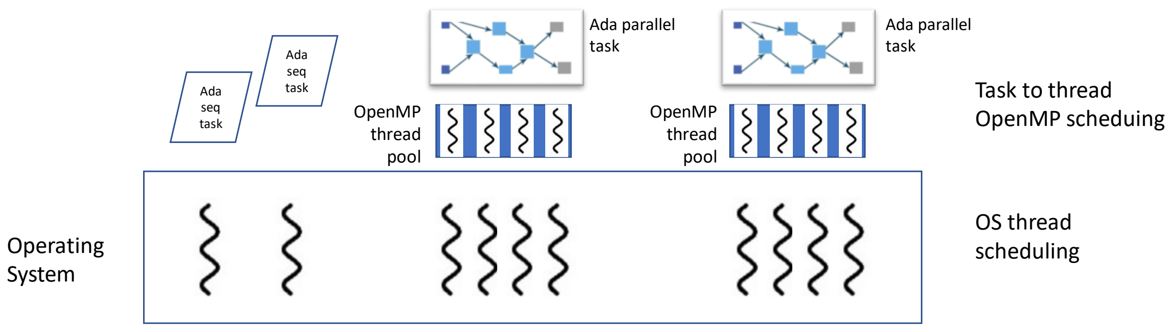


Figure 1. Model of an Ada application with 4 tasks: 2 with no parallel constructs, and 2 with parallel constructs.

Additionally, the presented approach assumes that each OpenMP thread is mapped 1-to-1 to operating system threads, which is typically the case (e.g., GCC's libgomp) since threads are reused only when there is no nested parallelism and between consecutive parallel regions. This mapping allows to simplify analysis and implementation (it makes an Ada task enclosed in a defined set of threads).

At the same time, several works have tackled the OpenMP standard considering Real-time restrictions in two directions: time predictability and functional safety. Regarding the former, the suitability of the OpenMP tasking model to derive timing guarantees based on a Task Dependency Graph (TDG) has been proved [12], the timing behaviour of both tied and untied tasks has been characterized [13] [14], and the OpenMP specification has been analysed and provided with augmentations to support the features needed in critical real-time systems [8]. Regarding the latter, the functional safety of the OpenMP 4.5 specification has also been analysed [15], and several works tackle different aspects such as correctness, including race conditions [16] and deadlocks [17], resiliency [18] and programmability [19] [20].

4 Issues which have been addressed

The impact of fine-grained parallelism in the Ada specification has been discussed for several years, in particular in the context of the IRTAW Workshop [4]. Several of the issues which have been discussed are already addressed in the proposed Ada 2020X standard [1].

Parallelism inside protected actions

One of the recurrent issues was what would happen in parallel code would be executed whilst the Ada task was executing a protected action. This has been addressed, and the solution is to force that all logical threads of control created inside a protected action are executed (in an arbitrary order) by the logical thread of control which is executing the protected action (LRM 9.5.1) [1]. This, in fact, makes the execution sequential.

Potentially blocking operations

The execution of a potentially blocking operation inside parallel code was a subject of intense discussion, and one of the main challenges in supporting fine-grained parallelism. Although a solution was proposed [21], the adopted approach was to disallow these operations, and handle this situation as a bounded error (LRM5.1-18/5) [1]. This makes use of the new Nonblocking aspect in Ada 202X.

Abort and transfer of control

Another issue which was discussed several times was what would happen in the case of task cancelling. The approach (LRM 5.1-16/5 and 17/5) [1] is to attempt to cancel all logical threads of control, and prevent new ones to be started. Note that the specification allows deferring the cancel further than an abort deferred operation, but not the creation of new parallel constructs. Parallel constructs are not abort-deferred operation, but abort must be made as soon as an abort completion point is reached (LRM 9.8) [1]: for parallel constructs this is where a new logical thread of control would be created or the end of the parallel construct. Both Ada and OpenMP allow for cancelation to be checked in specific predefined points (OpenMP parallel regions can only be cancelled by means of cancellation constructs, i.e., cancel and cancellation point). Hence, in order to be able to stop the parallel computation in an OpenMP region the compiler can insert cancellation points in places where it is safe to abort the operation without affecting the correctness of the application.

Shared variables

The parallel access to shared variables is addressed both considering the rules for unsynchronized access to variables (LRM 9.10) [1], and access to Atomic and Volatile objects (LRM C.6) [1]. Ada already had the notion of concurrent activities accessing shared data, which were only necessary to be updated to consider logical threads of control instead of tasks. Note that this also includes access to task attributes, which were already required to use locks (LRM C7.2) [1].

Exceptions

From the point of view of the model presented in this paper, exceptions are in fact handled as a transfer of control issue. If some exception is raised and handled inside the code being executed in one OpenMP thread, then there is no impact, as it is handled inside this thread. If, nevertheless, it is propagated outside of the parallel construct, it is necessary to cancel the execution in the other threads (which is made as noted for the transfer of control). If several exceptions are raised, an arbitrary one can be selected (LRM 5.1 16/5) [1] (the AdaCore implementation selects the "first" that informs the runtime [5]).

5 Real-time systems

Real-time systems impose different requirements regarding the scheduling, which are also affected by the use of parallel regions inside tasks.

Task priorities

The priority model which is specified in Annex D (LRM D.1) [1] specifies that each Ada task is provided with a Priority aspect that defines a degree of urgency, and that, except when explicitly noted, should be used to determine the next task to execute: processors are allocated to the ready task with the highest priority value. OpenMP also has a notion of priority, but that is only used as a hint, and a recommendation for the execution of OpenMP tasks (OpenMP specification, 2.10.1) [22].

Nevertheless, the current model of mapping OpenMP threads 1-to-1 to operating system threads, allows to address this difference, if the priority of the Ada task is propagated to the underlying OS threads. In this case, OpenMP priorities are hidden due to the black-box nature of task-to-thread mapping. All threads from the same thread pool would have the same priority (the priority of the Ada task), thus the semantics of Ada priority would be kept, and analysis is possible to guarantee real-time requirements.

It is nevertheless important to note that there are works proposing that OpenMP task priorities are made global to all thread pools, and that priority becomes more than just a recommendation [8]. But in this case, it is possible to propagate the Ada task priorities to OpenMP tasks priorities, thus still maintaining the semantics of Ada priorities.

Dynamic changes to priorities

Priority in Ada may change in two different conditions (LRM D.1 19/3-24) [1]:

- The base priority of a task is changed through a Set_Priority procedure call.
- The active priority of a task changes due to priority inheritance (e.g., when accessing a protected object).

Either changes to the base priority or active priority must be taken into consideration for the execution in a parallel construct. This can happen, for instance, if a task changes its priority (or has its priority changed) with Set_Priority while executing parallel code (thus, actually executing in more than one thread), or if one of the "branches" of the parallel construct executes a call to a protected object with a ceiling priority higher than the current active priority of the task.

For the case of changing the base priority of the task, with Set_Priority, the proposal is to extend the current specification so that the change of the priority is only performed outside of the parallel construct. Currently, in a single processor, Set_Priority needs to be executed as soon as the task is outside of a protected action. For multiprocessors, an implementation needs to document the conditions that may delay the change. Note that this is stronger than the current approach for deferring transfer of control, which cannot be delayed past the creation of new logical threads of control. In this case, it would need to be deferred completely to the end of the parallel construct. This can be specified in the Dynamic priorities section of the LRM (D.5.1).

Priority inheritance is used to guarantee the correctness of access to shared protected objects (other sources of priority inheritance, e.g., task activation or accepting entry calls, cannot exist when executing in a parallel construct). In this case, when accessing a protected object with e ceiling priority higher than its current active priority, the task inherits the ceiling priority of the object. Although this still needs careful analysis, a potential approach is to change only the active priority of the parallel "branch" executing the protected action, thus allowing a task to have more than one active priority at a time (replacing task by logical thread of control in section D.3 of the LRM, and allowing for multiple active priorities in section D.1). Note that currently a task only has one active priority, therefore if a logical thread of control accesses a protected object, all parallel logical threads need to change.

Other dispatching models

Ada specifies not only preemptive priority-driven scheduling, but other different models:

- Non-preemptive dispatching.
- Round-robin dispatching.
- Earliest Deadline First (EDF) dispatching.

For non-preemptive dispatching, as the Yield procedures are specified as Nonblocking => False, they cannot be called from within a parallel construct. Therefore, parallel code is always non-preemptible [2].

For Round-robin dispatching, the main issue is if a quantum is defined per task, or per logical thread of control. As it is currently specified, it is per task, which means that when the quantum is exhausted, all logical threads of control of that task need to also be moved to the tail of the ready queue. The main issue is how to account for the parallel execution time, and the accuracy of determining it has exhausted. This is similar to execution time timers, therefore the discussion is left for that section.

For EDF dispatching, the same approach as used for priorities can be used. If the underlying operating system supports EDF, the relative deadline of an Ada task needs to be propagated to the OS thread. OpenMP tasks have no deadlines, but again the mapping of OpenMP tasks to threads is treated as a black-box from the OS scheduling point of view.

Moreover, with the addition of parallel capabilities, other dispatching models may be interesting to explore:

- Limited preemptive dispatching: OpenMP executes OpenMP tasks in the threads non-preemptively. This is mainly to reduce preemption overhead, as well as reduce caching issues, as it is assumed that preempting while a computation is taking place increases the risk of invalidating cache lines being used by that computation. Due to that, a specific implementation of a lightweight OpenMP runtime [23] and OS [24] propagates the preemption points of OpenMP tasks to the underlying OS threads so that threads are only preempted at these points.
- For more dynamic parallel real-time systems, server-based scheduling is an interesting approach to support real-time applications. In this case, the need to account for execution budget becomes fundamental.

Execution time timers

Ada Annex D has several capabilities to handle execution time control (D.14), and in particular it allows to set handlers when a task or a group of tasks has used a specific

[2] Note that it is not explicit, in a multiprocessor setting, if Yield in one of the processors affects that processor only. If global scheduling is being used, there is only one dispatcher, but other processors may be executing non-preemptive tasks. Yield forces a task dispatching point, but it is implicit that only the processor where it is executed is affected.

amount of CPU time. When in a parallel setting, a task may be executing in more than one CPU at a time, therefore the implementation of an execution time timer needs to consider the execution time in all CPUs where the task is executing 3, which means that the detection of execution time expiration may not be immediate.

Several different solutions exist (e.g., to account only for execution time in the CPU where the task has set the handler, to disregard execution time accounting for parallel tasks, to consider that the execution time accounting is for each parallel thread independently). Nevertheless, budgets may be required to be per task, and in this case, the execution time needs to accumulate from different CPUs. This can be performed only at specific points in the parallel code (e.g., the same as used for cancelation polling), which means some loss of accuracy.

Asynchronous Task Control and Preemptive Abort

Preemptive Abort (LRM D.6) specifies that in a single processor an abort is completed immediately at the first point that is outside of an abort-deferred operation. This is more restrictive than for the general case (LRM 9.8), but possible in a single processor. For the multiprocessor case, the only requirement is that an implementation document any further delay.

The Asynchronous Task Control (LRM D.11) specifies the behaviour in terms of the notion of the *held* priority. Similar to inherited priorities, it is necessary to understand if this reflects in the logical thread of control, or the whole task. The latter case is potentially the preferred one, which means that if it is accepted that a task may have multiple active priorities (see Dynamic changes to priorities above), then the held priority must be made clear as affecting all active priorities of a task.

Multiprocessor dispatching domains

The notion of dispatching domains is used in Annex D to specify the set of processors where a task may execute. This allows to specify from a fully global scheduling (all tasks may execute in all processors with a single ready queue) to fully partitioned scheduling (each task is statically assigned to one particular CPU, and each CPU has a separate, even if conceptual, ready queue).

However, when going into a parallel setting, there is another dimension, which is the fact that an Ada task may be executing in several processors, but it may be relevant to disallow migrations between the processors (a computation which starts in a specific core always continues in this core). This model is applicable to the case where the Ada task has a set of threads available for the parallel execution, with each one of these threads pinned to one core. The current support for dispatching domains does not allow such model, since if a task is able to execute in more than one core, there is a single ready queue in the domain, and worker threads may migrate from core to core.

[3] This does not happen for group execution time budgets, since these are per CPU (precisely to avoid the multiprocessor accounting of execution time).

Therefore, a possibility is to extend the current model with a Logical_Thread_of_Control object added to the Assign_Task procedure, which will provide the information on the allocation of underlying threads per CPU, as well as if migration is allowed.

Control of the underlying runtime (number of OpenMP tasks and worker threads)

Although analysis exists that can extract an extended task dependency graph of a code with OpenMP tasking annotations [25], and that therefore can be applied to Ada parallel code, one important issue for real-time is the possibility to define the number of worker threads of a parallel construct, as well as the parallel load in each thread. For the latter, parallel blocks are not an issue (as the number of parallel "branches" is fixed), but in parallel loops it is necessary to specify the amount of parallelism to provide. The current draft of the standard allows to specify the maximum number of chunks (LRM 5.5) [1] of a loop, but not a minimum, or a specific number.

Concerning the former, although not in the draft standard, the current prototype implementation [5] allows to specify the number of threads a task will have available for parallelism. However, this will be a per Ada task number, which will be fixed, and no possibility exists to specify in a particular loop, iterator or block, the actual number of threads to be used.

Acknowledgements

This work was supported by the Spanish Ministry of Science and Innovation under contract TIN2015-65316-P and by the European Union's Horizon 2020 Research and Innovation Programme under grant agreements No 825473 and No 871669.

References

[1] Ada Rapporteur Group, "*Ada Reference Manual, 202x Edition, Draft 26*," 2020. [Online]. Available: http://www.ada-auth.org/standards/2xrm/html/RM-TTL.html. [Accessed September 2020].

[2] L. M. Pinho, B. Moore and S. Michell, "Parallelism in Ada: status and prospects" in *International Conference on Re-liable Software Technologies – Ada-Europe 2014*, LNCS 8454, Springer, 2014.

[3] S. Royuela, C. Martorell, E. Q. X and L. M. Pinho, "OpenMP tasking model for Ada: safety and correctness" in *22nd International Conference on Reliable Software Technologies (Ada-Europe 2017)*, pp 184-200. Vienna, Austria, 2017.

[4] L. M. Pinho and T. Vardanega, "Session Summary: Parallel Programming," in *IRTAW 2018*, Ada Lett. 38, 1, 58–60, 2018. DOI:https://doi.org/10.1145/3241950.3241960, 2018.

[5] S. T. Taft, "*Report on Ada 202X light-weight parallelism features*" 2020.

[6] B. Chapman, L. Huang, E. Biscondi, E. Stotzer, A. Shrivastava and A. Gatherer, "Implementing OpenMP on a High-Performance Embedded Multicore MPSoC," in *International Symposium on Parallel & Distributed Processing*, 2009.

[7] A. Marongiu, P. Burgio and L. Benini, "Supporting OpenMP on a Multi-cluster Embedded MPSoC," *Microprocessors and Microsystems*, vol. 35, no. 8, pp. 668-682, 2011.

[8] M. A. Serrano, S. Royuela and E. Quiñones, "Towards an OpenMP Specification for Critical Real-Time Systems," in *International Workshop on OpenMP* (IWOMP), 2018.

[9] M. Garcia, J. Corbalan, R. M. Badia and J. Labarta, "A Dynamic Load Balancing Approach with SMPSuperscalar and MPI," in *Facing the Multicore-Challenge II*, 2012.

[10] S. Royuela, M. A. Serrano, M. Garcia-Gasulla, S. M. Bellido, J. Labarta and E. Quiñones, "The Cooperative Parallel: A Discussion about Run-time Schedulers for Nested Parallelism," in *International Workshop on OpenMP* (IWOMP), 2019.

[11] S. Royuela, L. M. Pinho and E. Quiñones, "Enabling Ada and OpenMP Runtimes Interoperability through Template-based Execution," *Journal of Systems Architecture*, vol. 105, 2020.

[12] R. Vargas, E. Quiñones and A. Maronjiu, "OpenMP and Time Predictability: A Possible Union?" in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[13] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna and E. Quiñones, "Timing Characterization of OpenMP4 Tasking Model," in *International Conference on Compilers, Architecture and Synthesis for Embeded Systems (CASES)*, 157-166.

[14] J. Sun, N. Guan, Y. Wang, Q. He and W. Yi, "Real-time Scheduling and Analysis of OpenMP Task Systems With Tied Tasks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2017.

[15] S. Royuela, A. Durán, M. A. Serrano, E. Quiñones and X. Martorell, "A Functional Safety OpenMP* for Critical Real-time Embedded Systems," in *International Workshop on OpenMP (IWOMP)*, 2017.

[16] U. Banerjee, B. Bliss, Z. Ma and P. Petersen, "A Theory of Data Race Detection," in *Workshop on Parallel and Distributed Systems: Testing and Debugging*, 2006.

[17] D. Kroenig, D. Poetzel, P. Schrammel and B. Watcher, "Sound Static Analysis for C/Pthreads," in *IEEE/ACM International Conference on Automated Software Engineering*, 2016.

[18] M. Wong, M. Klemm, A. Duran, T. Mattson, G. Haab, B. R. de Supinski and A. Churbanov, "Towards an Error Model for OpenMP," in *International Workshop on OpenMP*, 2010.

[19] S. Royuela, A. Duran, C. Liao and D. J. Quinlan, "Auto-scoping for OpenMP Tasks," *in International Workshop on OpenMP (IWOMP)*, 2012.

[20] S. Royuela, A. Durán and X. Martorell, "Compiler Automatic Discovery of OmpSs Task Dependencies" 2012.

[21] L. M. Pinho, B. Moore, S. Michell and S. T. Taft, "An Execution Model for Fine-Grained Parallelism in Ada," in *Proceedings of the 20th Ada-Europe International Conference on Reliable Software Technologies*, Madrid Spain, June 22-26, 2015.

[22] OpenMP Architecture Review Board, "*OpenMP Application Programming Interface*" 2018.

[23] A. Marongiu, G. Tagliavini and E. Quiñones, "OpenMP Runtime," in *High-Performance and Time-Predictable Embedded Computing*, 2018.

[24] C. Scordino, E. Guidieri, B. Morelli, A. Marongiu, G. Tagliavini and P. Gai, "Embedded Operating Systems," in *High-Performance and Time-Predictable Embedded Computing*, 2018.

[25] M. A. Serrano, S. Royuela, A. Marongiu and E. Quinones, "Predictable Parallel Programming," in *High-Performance and Time-Predictable Embedded Computing*, 2018.

[26] M. A. Serrano, S. Royuela and E. Quiñones, "Towards an OpenMP Specification for Critical Real-time Systems," in *International Workshop on OpenMP (IWOMP)*, 2018.

[27] AdaCore, "*GitHub - AdaCore/gnat-llvm: LLVM based GNAT compiler*" 2020. [Online]. Available: https://github.com/AdaCore/gnat-llvm.

[28] S. Royuela, R. Ferrer, D. Caballero and X. Martorell, "Compiler analysis for OpenMP tasks correctness," in *International Conference on Computing Frontiers (CF)*, 2015.

[29] B. Moore, L. M. Pinho, S. Michell, "Tasklettes – a Fine Grained Parallelism for Ada on Multicores," in *International Conference on Reliable Software Technologies – Ada-Europe 2013*, LNCS 7896, Springer, 2013.

Vectorization Challenges in Digital Signal Processing *

Jorge Garrido, David Pisonero, Juan Zamorano, Juan A. de la Puente
Sistemas de Tiempo Real e Ingeniería de Servicios Telemáticos (STRAST)
Information Processing and Telecommunications Centre (IPTC)
Universidad Politécnica de Madrid (UPM); email: str@dit.upm.es

Abstract

The paper analyses the support for vectorization that can be found in some programming languages, and the ways it could also be used in Ada. A proposal for an Ada extension for enhanced vectorization support is included.

1 Introduction

The vector processing capabilities of current mainstream general-purpose CPUs enable signal processing applications to be implemented on low-cost platforms, thus making such systems easier to maintain and build from COTS components. Most vector processing architectures implement the SIMD (single instruction – multiple data) concept of parallelism, where an operation is performed on multiple array elements at the same time. A well-known example is the sets of Intel x86 MMX, SSE and AVX instructions [1].

Some common algorithms which can be efficiently implemented on SIMD architectures include Kalman filters [2, 3] and Fast Fourier Transforms (FFT) [4, 5], as well as many other algorithms commonly used in signal processing and control systems. Such algorithms are often used in embedded real-time systems with high integrity requirements, where Ada is the primary choice as a programming language. Therefore, it seems to be worth exploring the use of Ada on vector architectures.

In the rest of this paper we provide a preliminary view in that direction. Section 2 introduces the current support for vectorization in programming languages, including auto-vectorization, guided vectorization, and low-level vectorization, as well as some performance measurements for a simple example using these approaches. The support for vectorization in Ada is examined in section 3, and a proposal is made for an Ada extension providing further support for vectorization. Finally, conclusions are drawn in section 4.

*This work has been partially funded by the Spanish National R&D&I plan (project PRECON-I4, TIN2017-86520-C3-2-R).

2 Current vectorization support

Current vectorization alternatives and support can be categorized following different criteria. One of these criteria is the abstraction level.

- Auto vectorization: vectorization is fully controlled by the compiler, normally upon user request. gcc and icc support auto vectorization using the -O3 flag.
- Guided vectorization: the programmer provides partial information on where and how to vectorize with either pragmas or hints.
- Low level vectorization: the programmer directly calls vector instructions either via intrinsics or assembly instructions. C++ provides classes for manipulating common data types.

Alternatively, for certain applications such as signal processing there are libraries with optimized functions using vectorization, among other techniques. One example of this is the Intel® Math Kernel Library [6].

In the rest of this section a brief insight on relevant examples of each previously mentioned approach will be provided, along with an example based on a sum of two arrays. Finally, a performance comparison among presented approaches will be provided based on measured execution times of given examples.

2.1 Auto vectorization

Several compilers incorporate nowadays vectorization as part of the automatic optimization procedures. This approach eases the adoption of vectorization, in particular for legacy code or for inexperienced users. Its main drawbacks are the loss of control over the implementation and the time and space required for compilation, since, unless the processor microarchitecture is known a priori and declared using compiler switches, most compilers rely on generating different versions for the same function, including different vectorization alternatives.

Among the compilers supporting auto vectorization are gcc and icc which do so via the –O3 (gcc) and –O2 and –O3 (icc) flags. icc also includes a reporting feature, which provides the user with information about which sections of code have been vectorized, and which could be vectorized but present difficulties preventing the vectorization (such as data not properly aligned).

Listing 1 presents the example that will be used in the rest of the document. The code tests the performance of a sum of vectors. In this case as auto vectorization is used, this presented code is plain C++ and results obtained compiling with gcc and −O2 flag will serve as a reference for comparison purposes, while results with −O3 (and icc −O2) will showcase the speedup using vectorization.

Listing 1: Running example implementation in plain C++.

```
float[] add(float vinA[], float vinB[],
  int count) {
    vout[count];
    for(int index = 0; index < count; index++) {
        vout[index] = vinA[index] + vinB[index] ;
    }
    return vout;
}
```

2.2 Guided vectorization

Guided vectorization provides the user with higher control over the vectorization optimization. This control is expressed via pragmas or specific syntax, including reserved words and extended array notation. Languages that natively incorporate guided vectorization include Julia or R. C inspired languages also providing means for vectorization are ISPC and OpenCL. The former has the advantage of being designed to be integrated into C-based programs by just linking with the generated .o files. The latter, while also based in C, uses a Just-In-Time compilation mechanism to provide both task and data parallelism as well as native support for GPU allocation of chunks.

Listing 2 presents the running example implementation of the vector sum using ISPC. This code, as mentioned before, is compiled using the ispc compiler to produce a .o file to be later linked with the program accessing the function. In this code, the foreach statement provides the compiler the domain of integer range on which the function is going to execute. It can include a multi-dimensional domain separated by commas (eg: foreach (j = 0 ... height, i = 0 ... width)), but can not contain break statements.

Listing 2: Example implementation using Intel® ISPC.

```
export void add(uniform float vinA[],
 uniform float vinB[], uniform float vout[],
  uniform int count) {
    foreach (index = 0 ... count) {
    // Load the appropriate input value for this program instance.
        float vA = vinA[index];
        float vB = vinB[index];

        // And write the result to the output array.
        vout[index] = vA + vB;
    }
}
```

Another alternative from Intel® is Cilk™. For what regards to vectorization, it follows what its called the array notation, providing explicit data parallelism.

Listing 3: Example implementation using Intel® Cilk™array notation.

```
float[] add(float vinA[], float vinB[],
  int count) {
    float vout[count];
    vout[0:count] = vinA[0:count] + vinB[0:count];
    return vout;
}
// or just C[:] = A[:] + B[:]; if A and B are statically allocated
```

Finally, both the Intel® icc compiler and OpenMP accept vectorization hints from the user in the form of pragmas [7,8]. Furthermore, the latter allows combining vectorization and parallel instructions in for loops.

2.3 Low level vectorization

Finally, Intel® also does offer the possibility of having full control over the vectorization via intrinsics [9]. Intrinsics, provide access to vectorized functions and data types. This includes different (power of two-length) data types and basic functions for arithmetics, bit manipulation, math functions, cryptography, among others.

Listing 4: Example implementation using Intel® intrinsics.

```
float[] add(float vinA[], float vinB[],
  int count) {
    float vout[count];
        for (int i=0; i<count; i+=16){
         __mm512 Avec = _mm512_load_ps (vinA+i);
         __mm512 Bvec = _mm512_load_ps (vinB+i);
         Avec = _mm512_add_ps (Avec,Bvec);
         _mm512_store_ps(vout+i, Avec);
     }
     return vout;
}
```

A more legible code than that of Listing4 can be achieved using the C++ vector class library [10].

Listing 5: Example implementation using C++ vector class library.

```
float[] add(float vinA[], float vinB[],
  int count) {
    float vout[count];
        for (int i=0; i<count; i+=16){
         F32vec16 *Avec = (F32vec16*) (vinA+i);
         F32vec16 *Bvec = (F32vec16*) (vinB+i);
         F32vec16 *Ovec = (F32vec16*) (vout+i);
         *Ovec = *Avec + *Bvec;
     }
     return vout;
}
```

Main drawback of this approach is that the user is fully responsible for the correctness and efficiency of the solution. In particular, the user is responsible for the "peel loop" (iterations until data is aligned) and "remainder loop" for data exceeding power of two size.

2.4 Performance

Table 1 presents the average execution time of the example with a relevant number of elements under different configurations. As can be seen, the use of SIMD instructions produces a notable improvement as it halves the execution time for the example. Among vectorized approaches, performance-specific guided vectorization of ISPC provides roughly a 10% improvement over the auto vectorization approaches. Intrinsics provide an extra ∼3% of execution time reduction.

3 Vectorization in Ada

3.1 Current support for vectorization in Ada programs

Ada applications can nowadays benefit from vector instructions. This can be achieved following different approaches. One approach is to link the Ada application with specific pieces of code developed with languages and tools providing guided or low-level vectorization using Ada interfacing functionality. This has been done importing and linking the ISPC vector function in Listing 2, producing the results shown in Table 2. The other potential approach is to rely on the compiler to automatically produce vector instructions when appropriate flags are passed. Results of a native Ada implementation compiled with gnat 8 and −O3 flag (and −O2) for comparison are also presented in Table 2. Finally, gnat includes an implementation defined **pragma** Loop_Optimize which accepts Ivdep | No_Unroll | Unroll | No_Vector | Vector as optimization hints. For this **pragma** to take effect, relevant switches (−funroll−loops −ftree−vectorize) are still required.

As can be seen, measured times are similar to those of Table 1 with Ada vectorization (both automatic with the −O3 flag and using the pragma), producing slightly worst results, particularly when the number of elements is not a power of two. It does, however, also roughly halve the execution time of vectorized code.

Despite the execution time improvement resulting of these approaches, none of them match Ada standards in the main strengths of the language in its application domains. Importing code generated with vector-specific languages and compilers increases the number of code pieces and tools subject to verification and certification. Just relying on auto vectorization undermines user influence on performance and low-level, potentially platform-based, control of execution. Finally, using a compiler defined pragma for requesting the vectorization itself can cause portability issues and limits the expressiveness to provide more refined vector optimization hints.

3.2 Proposal

Following the adoption of the parallel execution model and parallel loop, the language can also benefit from vectorization support. In particular, and following the parallel model, a native guided vectorization approach could be aligned with the language design goals and provide sufficient support for Ada traditional areas of application, such as signal processing.

We propose the reserved word vector to be used at the iteration scheme of for loop statements to indicate the user request for the use of SIMD instructions inside the loop when available.

Listing 6: Example implementation using Ada vector proposal.

```
vector
for I in V_Out'Range loop
  V_Out (I) := V_In_A(I) + V_In_B(I);
end loop;
```

Same as OpenCL and OpenMP, a for loop could also be both paralleled and vectorized, potentially defining a number of chunks. parallel precedes vector since what can be vectorized is the data processed by each chunk.

Listing 7: Example implementation using Ada vector proposal and parallel notation.

```
declare
  subtype Chunk_Number is Natural range 1 .. 8;
begin
   parallel (Chunk in Chunk_Number) vector
   for I in V_Out'Range loop
      V_Out (I) := V_In_A(I) + V_In_B(I);
   end loop;
end;
```

Control over vectorization could be refined by means of implementation defined pragmas. In contrast to current gnat implementation, icc and/or OpenMP include hints on the average, maximum, minimum or exact number of iterations, vector length, if remainders are to be also vectorized or not, or to produce an error at compile time if the loop can not be vectorized among others. These optimization parameters are essential to produce efficient implementations of math-intensive applications such as real-time FFT-based signal processing and could foster the development of new application-specific high-performance Ada libraries .

Table 1: Summary of average time executions of running example. Values are average of 100,000 executions and are expressed in μs.

Test	GCC -O2	GCC -O3	ICC -O2	ICC -O3	ISPC	Intrinsics
500.000 elements	260	113	111	110	102	100
524.288 (2^{19}) elements	275	125	122	118	111	108

Table 2: Summary of average time executions of Ada-based example implementations. Values are average of 100,000 executions and are expressed in μs.

Test	Ada -O2	Ada ISPC	Ada vectorization
500.000 elements	249	107	126
524.288 (2^{19}) elements	351	129	129

4 Conclusions

The Ada language is well known for its rich tasking model, strong typing and native support for many low level programming features. Recently, the language has benefited from the adoption of multiprocessor platforms and related updates in the tasking model. Next language revision will also incorporate parallel execution of instructions within a **for loop**.

In this paper the state of the art with regards to parallel execution of instructions over array-stored data, taking advance of vector instructions present in modern processors, is reviewed. This review is enriched with an example highlighting the performance increase that can be obtained using these instructions. Then, current Ada support for vector instructions is analyzed, finding potential areas of improvement. Finally, a language extension is proposed to enhance and formalize this support while being consistent with the adopted parallel model.

References

[1] H. Amiri and A. Shahbahrami, "SIMD programming using Intel vector extensions," *Journal of Parallel and Distributed Computing*, vol. 135, pp. 83 – 100, 2020.

[2] R. E. Kalman, "A new approach to linear filtering and prediction problems," *ASME Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[3] S. Gorbunov, U. Kebschull, I. Kisel, V. Lindenstruth, and W. Müller, "Fast SIMDized Kalman filter based track fit," *Computer Physics Communications*, vol. 178, no. 5, pp. 374–383, 2008.

[4] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[5] J. W. Cooley, P. A. Lewis, and P. D. Welch, "The fast Fourier transform and its applications," *IEEE Transactions on Education*, vol. 12, no. 1, pp. 27–34, 1969.

[6] "Intel math kernel library." `https://software.intel.com/en-us/mkl`. Accessed: 2020-02-04.

[7] "Intel® C++ compiler 19.1 developer guide and reference - intel-specific Pragma Reference - SIMD." `https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference-simd`. Accessed: 2020-02-04.

[8] "OpenMP 5.0 API Specification - 2.9.3 SIMD Directives." `https://www.openmp.org/spec-html/5.0/openmpsu42.html`. Accessed: 2020-02-04.

[9] "Intel® intrinsics guide." `https://software.intel.com/sites/landingpage/IntrinsicsGuide`. Accessed: 2020-02-04.

[10] "Intel® C++ compiler 19.1 developer guide and reference - C++ classes and SIMD operations." `https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference-c-classes-and-simd-operations`. Accessed: 2020-02-04.

The Problem of the Nested Squares

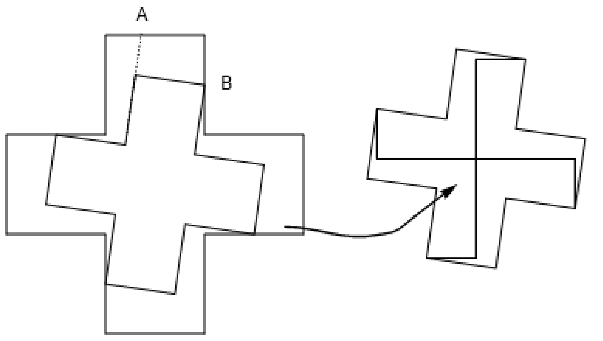
John Barnes

11 Albert Road, Caversham, Reading, RG4 7AN, UK; Tel: +44 118 9474125; email: john@jbinformatics.co.uk

Hello readers

We start by considering the solution to the problem about a Greek Cross given last time.

The problem was to divide a Greek cross into two equal small crosses as in the diagram below. The question is where are the points *A* and *B*? The Boys' Own Paper for January 1918 said that *B* is at the midpoint of the short side, which is correct. It also claimed that *A* is one-third of the way along the end. However, that is not correct. So where is *A*?

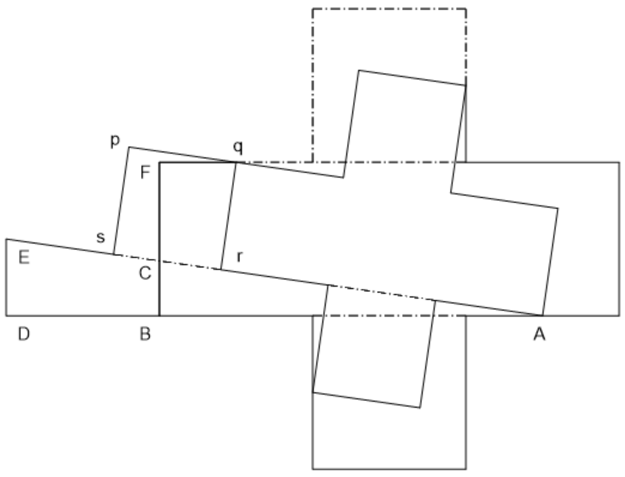


There are various ways of tackling this using a dash of tiresome trigonometry. They often use the formula for the tangent of the sum of two angles which one might (or might not) remember is

$$\tan(A + B) = (\tan A + \tan B) / (1 - \tan A \times \tan B)$$

But there is a cunning trick involving imagining cutting a lump off the diagram and moving it to a different place.

Thus we take the quarter part from the top, rotate it, and then stick it on the left end to give the diagram below:



It fits because *q* is exactly half way along the side edge of the big cross. So *pqrs* is a square and *ACE* is a straight line.

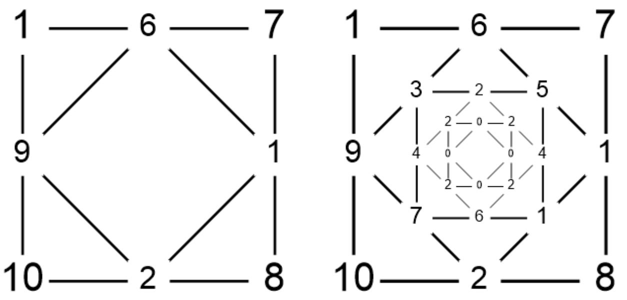
Now the triangles *ABC* and *ADE* are similar. Therefore

$$BC = AB \times DE / AD$$

If the side of the original cross is $2a$, we see that $AB = 5a$, $AD = 7a$, and $DE = a$. Hence $BC = 5a/7$ and therefore the point *C* is 5/14 of the distance from *B* to *F*. Remember that the BOP said one third which is 5/15. A near miss!

The next puzzle is about integers and squares. I mentioned this at the Ada-Europe conference in Stockholm in 2012 where the banquet was in a market. Gosh that was nearly 10 years ago.

Take a square and put an integer at each corner. Now put the difference between the integers on each side in the middle of the side and join the middles of the sides to together. We now have an inner small square with integers at each corner. Repeat the process until we have zero at all the corners.



The diagram shows the situation after the first inner square is drawn and the final position with all zeroes.

In this example it took five iterations to get all zeroes. The question is what is the maximum number of iterations required? It is easier to contemplate if we write them out as follows

```
1       7       8       10
    6       1       2       9
3       5       1       7
    2       4       6       4
2       2       2       2
    0       0       0       0
```

This curious problem is not as easy as it might look. I am grateful to my old college friend Rodney Baxter of the University of Canberra for interesting discussions on this and related matters.

An auxiliary question is what has the following cubic equation to do with this puzzle?

$$x^3 - 3x^2 - x - 1 = 0$$

and what are the roots?

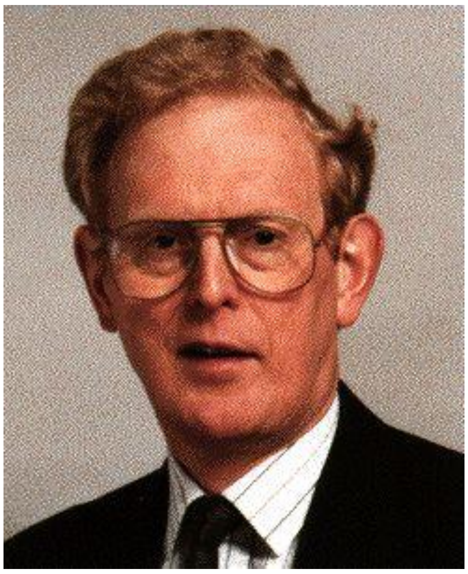
In memoriam: Ian Christopher Wand

Ian C Wand, Computer Scientist, was born on 10 December 1941 and died on 17 July 2020.

His team wrote the York Ada compiler. Under his leadership, the Computer Science Department at York matured, and gained international recognition.

Later he served as Deputy Vice-Chancellor of the University, playing an important part in setting up the Hull York Medical School.

He retired in 2002. Ian Wand is survived by his wife, Helen, children Paul and Celia, and their families.



Ian Wand was an all-round Computer Scientist, who, like all of his generation, came from another subject: in his case, Physics. He was born in Lincolnshire in 1941, studied Physics at the University of Leicester, obtaining his PhD in 1962. In 1965, he became a Lecturer there in Radio Physics, with an interest in ionospheric propagation, which led him to discover the fascination of computers and to start in the new subject.

His central interest was the study of Programming Languages with their compilers and programming support systems, and the discipline of programming, that became the subject now known as Software Engineering. He started working in computing in 1969 at the IBM Research Laboratories at Hursley in Hampshire and Yorktown Heights in the USA. He started on PL/I (IBM's new programming language) but also learned about the newest international programming language of the time, Algol-68. He found that a compiler for Algol-68 was being developed at Cambridge.

He was appointed as a lecturer at York in 1972, and brought a copy of that Algol-68C compiler which he adapted for the Elliott (later ICL) 4130 here. The compiler generated code for a virtual machine, so the transfer involved porting the compiler itself (written in BCPL) and also converting the virtual machine code to 4130 machine code.

By present standards, the start of the Department of Computer Science at York was extraordinary – originally set up to provide the Computing Service for the whole University, the department initially taught only Computation as a subsidiary subject, but after becoming Computer Science in 1973 progressively expanded its coverage to teach the whole subject and begin research activities. (This was in spite of a general feeling, even at the highest levels of the University, that the study of computers was but one step removed from the study of lawnmowers.) Ian, having already established himself as a successful lecturer and researcher, immediately extended his interest to other programming languages and their compilers.

Thus, when the focus of the Department moved to Real-Time Systems, he enthusiastically took up work on Modula (by N. Wirth) and, with a research grant from the SERC in 1975 entitled “Real-time programming language” led a team that made a compiler for Modula on the PDP-11. As the department grew, Ian was central in developing the syllabus for a full Computer Science degree, and in expanding the research on Real-Time Systems. He also excelled in managing the research staff.

As a result of his contributions to European studies of real-time programming, Ian was invited by Jean Ichbiah to comment on the developing Green language, and was awarded a major research grant in 1980 to study “Computing facilities for research in software technology” – which meant writing a compiler for Ada! (Ian Wand is recorded in the 1983 Ada manual as one of many people having had a constructive influence on the language). The York Ada Compiler (formally the University of York Ada Workbench Compiler System, YAWCS), was validated on 18/19 August 1986 and was the only one of its kind produced entirely within the UK.

With the benefit of his extensive understanding of Software Engineering, the compiler Wand's team produced [J. R. Firth, C. H. Forsyth, & I. C. Wand: “The Compilation of Ada”, Software – Practice and Experience, Volume 26, Issue 8, Pages 863-909, August 1996] was small, and easy to move from machine to machine. It can compile very large programs, involving many units and libraries, and integrates well with the UNIX environment. The compiler was written in C, and has been ported to six different architectures and validated under four different versions of the Validation Test Suite; it has been used widely academically and is sold commercially. The principal technical innovations of the York Ada compiler were the use of software tools to construct some of the syntax-based parts of the system, and use of a machine description and tree automata to support portable code generation. Colin Runciman recalls that the possibility of rewriting the compiler in Ada was considered, but not pursued. By 1982, early versions of the compiler were used by other academic teams as a platform for experimental research in concurrent and distributed systems, also to support intensive courses in the novel aspects of programming in Ada.

In 1983, when the department was split to separate the conflicting interests of teaching the subject and providing computing facilities for the rest of the university, Ian Wand was ideally placed to take over as Head of Computer Science, so that Ian Pyle could concentrate on the Computing Service. Ian Wand proved to be a superb administrator, and as Head of Department, he nurtured the process of building the Department as it grew to maturity, with the size, scale and reputation it enjoys today, from six to more than thirty teaching staff. He was appointed to a personal chair in 1984. Ian was also the founder and course director of the Department's advanced MSc courses in Safety Critical Systems Engineering (which involved lectures on Spark) in 1994, and in Software Engineering in 1997.

He understood immediately the importance of a strong research environment (even before research assessment exercises, for which he would later become a superb panel chair, were envisaged), and used the Department's research reputation to attract large numbers of well-taught, able students. He was able to navigate the world of the University without any desire for self-aggrandisement, but through the deployment of logical arguments designed to attract more staff, students, money and reputation to the benefit of the subject and Department he loved. Ian was instrumental in raising its level of research from gleams in the eye to its present eminence, and was a member of many committees of SERC, the DTI, IEE and BCS, representing the Department, the University, or the discipline of Computer Science in the UK.

The Department's top rating in the Research Assessment Exercises (of 1996 and 2001) owes much to Ian's early and continuing efforts and leadership.

When no longer Head of Department in 1992, Ian spent a year at the Joint Research Centre of the European Communities, ISPRA, Italy. He made two visits to Japan to evaluate Japanese Software Engineering (1990 and 1994).

As Deputy Vice Chancellor – part-time at that point – his many contributions included leading research selectivity exercises across campus, investigating forensically the IAAS, and both helping to establish and then implementing the successful plan for the Hull York Medical School. After seven years as Pro and then Deputy Vice-Chancellor, Ian retired from the University, finally and completely, in September 2002. In November 2002, he was awarded an Emeritus Chair by the Senate of the University of York. He continued to live in York for several years but then moved to Woking for family reasons.

Ian married Helen (née Parkinson) in 1965. Helen was a teacher of English. They have two children Paul and Celia, who were brought up in York and now have their own families: Paul and Alison have four children: Alexander, Eleanor (Ellie), Adam and Alice; Celia and Darien have three boys: Max, Sam and Joe.

Ian and Helen visited Italy frequently, to keep in touch with the friends they made there; they took their summer holidays in Orkney by way of contrast and to escape global warming.

Ian was a great colleague, always involved, interested, wise, caring, compassionate and dedicated to the University; and his love of good music was proverbial. Ian (piano), Paul (piano) and Celia (violin) were all excellent amateur musicians, so one must not see Ian as having nothing to do when he was not doing a little light consultancy to keep his hand in. Even in retirement Ian retained a keen interest in the fortunes of Computer Science as an academic discipline, the health of the university sector as a whole, and York in particular. He was never partisan, but only interested in understanding the logic of any situation and its consequences.

By coincidence, Ian Wand's birthday was the same as that of Ada Lovelace (you couldn't make it up!). The end came in 2020, not caused by the Corona virus, although arrangements were hindered by it. Ian Wand was diagnosed with Amyloidosis in March, and treated with chemotherapy, but died on July 17. The funeral was limited to his close family (Paul was unable to travel from Canada) and held on August 5.

Ian Wand is remembered with appreciation and affection by his colleagues. He was a rare and special person, a wonderful colleague and true friend. We shall always remember him. It is hoped to hold a commemoration for his life and work when Covid permits. His legacy is the successful and thriving Computer Science Department at the University of York.

Ian C Pyle (editor), Bill Freeman, Neil Audsley, Helen Byard, John Barnes, Colin Runciman and Sir Ron Cooke.

Contact Ian Pyle at: St Giles Road, Skelton, York YO30 1XR; email: ian.pyle@cantab.net

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o KU Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Hubert B. Keller
Karlsruher Institut für Technologie (KIT)
Institut für Angewandte Informatik (IAI)
Campus Nord, Gebäude 445, Raum 243
Postfach 3640
76021 Karlsruhe
Germany
Email: Hubert.Keller@kit.edu
URL: ada-deutschland.de

Ada-France

attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada-Switzerland

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch

Ada-Europe Sponsors

Ada Edge

27 Rue Rasson
B-1030 Brussels, Belgium
Contact:Ludovic Brenta
ludovic@ludovic-brenta.org

AdaCore
The GNAT Pro Company

46 Rue d'Amsterdam
F-75009 Paris, France
Contact: Jamie Ayre
sales@adacore.com
www.adacore.com

ADA LOG

2 Rue Docteur Lombard
92441 Issy-les-Moulineaux Cedex
France
Contact: Jean-Pierre Rosen
rosen@adalog.fr
www.adalog.fr/en/

altran

22 St. Lawrence Street
Southgate
Bath BA1 1AN, United Kingdom
Contact: Stuart Matthews
sparkinfo@altran.com
www.altran.co.uk

Deep Blue Capital

Jacob Bontiusplaats 9
1018 LL Amsterdam
The Netherlands
Contact: Wido te Brake
wido.tebrake@deepbluecap.com
www.deepbluecap.com

Ellidiss Technologies

24 Quai de la Douane
29200 Brest, Brittany
France
Contact: Pierre Dissaux
pierre.dissaux@ellidiss.com
www.ellidiss.com

KONAD
Software for Control and Administration

In der Reiss 5
D-79232 March-Buchheim
Germany
Contact: Frank Piron
info@konad.de
www.konad.de

PTC®
Developer Tools

3271 Valley Centre Drive,
Suite 300
San Diego, CA 92069, USA
Contact: Shawn Fanning
sfanning@ptc.com
www.ptc.com/developer-tools

SYSADA

Signal Business Centre
2 Innotec Drive, Bangor
North Down BT19 7PD
Northern Ireland, UK
enquiries@sysada.co.uk
www.sysada.co.uk

systerel
Safe real-time solutions

1090 Rue René Descartes
13100 Aix en Provence, France
Contact: Patricia Langle
patricia.langle@systerel.fr
www.systerel.fr/en/

Tid rum

Tiirasaarentie 32
FI 00200 Helsinki, Finland
Contact: Niklas Holsti
niklas.holsti@tidorum.fi
www.tidorum.fi

VECTOR

Millennium Tower, floor 41
Handelskai 94-96
A-1200 Austria
Contact: Massimo Bombino
sales@at.vector.com
www.vector.com

WhiteElephant

Beckengässchen 1
8200 Schaffhausen
Switzerland
Contact: Ahlan Marriott
admin@white-elephant.ch

XGC Technology

United Kingdom
Contact: Chris Nettleton
nettelton@xgc.com
www.xgc.com

Ada europe