ADA USER JOURNAL

Volume 37
Number 2
June 2016

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

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

This second issue of 2016 was finalized shortly after the 21st International Conference on Reliable Software Technologies – Ada-Europe 2016, which took place in the beautiful scenario of the Scuola Superiore Sant'Anna, in Pisa, Italy. As usual, the conference week provided a very successful program of technical presentations, as well as a fruitful networking environment. I would like to point out not only the scientific program of the conference, as usual with a set of high-quality papers, but also the industrial track of the conference, with a set of high quality presentations, the tutorials provided on Monday and Friday, and the co-located DeCPS workshop also on Friday. The conference also featured three very interesting keynote talks, from Alan Burns, Professor at the University of York, UK, on programming languages for future cyber physical systems; Guido Ghisio, responsible for Automated Driving Technologies at Magneti Marelli, Italy, on challenges for the automotive platform of the future; and Marc Duranton, senior member of CEA, France, on the HiPEAC (European Network on High Performance and Embedded Architecture and Compilation) vision. A very successful exhibit during the central days of the conference allowed participants to learn the most recent developments in the commercial tools for the development of reliable software.

An extra highlight of this year's conference, was a Special Session on Ada and Parallelism. As put forward in the conference program:

Ada has been a language which has always excelled with its advanced high-level concurrency support. In the last 20 years, Ada has steadily extended its wealth of concurrency features and capabilities to a considerable extent, yet within the bounds of a sequential task reasoning. With the advances in processor, and in particular the move into a parallel world, it is time to discuss how Ada should be evolved into supporting in the language the notion of fine-grained parallelism.

The session included two presentations from Tucker Taft, of AdaCore, USA, and Brad Moore, of General Dynamics, Canada, with insights on potential approaches to support fine-grained parallelism in Ada, and an open discussion moderated by Jeff Cousins, of BAE Systems, UK and ARG rapporteur.

In this issue of the Journal we report on this session, starting with a summary of the discussion, by Pat Rogers, of AdaCore, USA, followed by three papers related to the presentations. The first paper, by Tucker Taft, provides a proposal for container iterators supporting parallelism, whilst in the second paper Brad Moore provides an approach based on the Paraffin library. Finally, the last paper, also by Brad Moore, describes how parallel reductions are handled in Paraffin.

The issue also includes a report on the recent 18th International Real-Time Ada Workshop, which took place last April in the beautiful scenario of Benicàssim, near Valencia, Spain. As usual, the workshop discussed proposals for real-time and high-integrity systems support in Ada. In a forthcoming issue of the Journal we will provide the more detailed summaries of the workshop sessions discussions.

Finally, the reader will find the News and Calendar and Events sections, by Jacob Sparre Anderson and Dirk Craeynest, their respective editors. A special note to the preliminary call for papers for the 22nd International Conference on Reliable Software Technologies – Ada-Europe 2017 that will take place June 2017 in Vienna, Austria.

Luís Miguel Pinho
Porto
June 2016
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Jacob Sparre Andersen

Jacob Sparre Andersen Research & Innovation. Email: jacob@jacob-sparre.dk

Contents

Ada-related Events

[To give an idea about the many Ada-related events organised by local groups, some information is included here. If you are organising such an event feel free to inform us as soon as possible. If you attended one please consider writing a small report for the Ada User Journal. —sparre]

CFP: SIGAda HILT 2016

From: S. Tucker Taft, AdaCore
Date: Mon, 9 May 2016 18:25:50 -0700
Subject: CFP: SIGAda HILT 2016 Workshop at ESWEEK on Models + Contracts; due June 30
Newsgroups: comp.lang.ada
Subject: CFP: HILT 2016 Workshop at ESWEEK on Models + Contracts; due June 30

Call for papers and extended abstracts:

HILT 2016 Workshop on Model-Based Development and Contract-Based Programming

As part of ESWEEK, October 6 & 7, 2016, Pittsburgh, PA

Sponsored by ACM SIGAda

CFP:
http://www.sigada.org/conf/hilt2016/HILT2016-CFP.pdf

Website: http://sigada.org/conf/hilt2016

ESWEEK: http://esweek.org

The High Integrity Language Technology (HILT) 2016 Workshop is focused on the synergy between Model-Based Development and Contract-Based Programming, producing a formal model-driven approach to the development of high-assurance software-intensive systems.

Important Dates:

June 30: Papers or Extended abstracts due

July 31: Notification of submissions accepted for presentation

Sep 15: Final submissions due

Oct 6&7: Workshop as part of ESWEEK

Keynote:

- Phil Koopman, CMU

We encourage papers and extended abstracts relating to:

- Architecture-level and requirements-oriented modeling with systems such as AADL, SysML, and ArgoSim
- Component-level modeling with systems such as UML/OCL, Simulink, and SCADE
- Automated analysis and code generation targeting verification-oriented tools and/or programming language subsets such as Coq, PVS, ACL2, Why, SPARK/Ada, Frama C/ACSL, MISRA C, JML, and CompCert C.
- Other contributions linking modeling and contracts to the topics associated with the co-located EMSOFT conference:
  - o Formal modeling and verification
  - o Testing, validation, and certification
  - o Model- and component-based software design and analysis
  - o Software technologies for safety-critical and mixed-critical systems
  - o Robust implementation of control systems
  - o Embedded software security

Workshop Co-Chairs

- Julien Delange, Software Engineering Institute
- Tucker Taft, AdaCore, Inc

Ada-Europe 2016 in Pisa

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Sun, 5 Jun 2016 20:46:39 -0000
Subject: Press Release - Reliable Software Technologies, Ada-Europe 2016
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

FINAL Call for Participation

*** UPDATED Program Summary ***

21st International Conference on Reliable Software Technologies
Ada-Europe 2016

13-17 June 2016, Pisa, Italy

http://www.ada-europe.org/conference2016

** Check out tutorials and workshop! **

***Full Program available on conference web site ***

* Printed proceedings available at event *

*** Register now! ***

Press release:

21st Ada-Europe Conference on Reliable Software Technologies

International experts meet in Pisa

Pisa (6 June 2016) - Scuola Superiore Sant'Anna and Ada-Europe organize from 13 to 17 June 2016 the "21st International Conference on Reliable Software Technologies - Ada-Europe 2016" in Pisa, Italy. The event is organized in cooperation with the Ada Resource Association (ARA), and with ACM's Special Interest Groups on Ada (SIGAda), on Embedded Systems (SIGBED), and on Programming Languages (SIGPLAN).

The Ada-Europe series of conferences has over the years become a leading international forum for providers, practitioners and researchers in reliable software technologies. These events highlight the increased relevance of Ada in general and in safety- and security-critical systems in particular, and provide a unique opportunity for interaction and collaboration between academics and industrial practitioners.

This year's conference offers two days of parallel tutorials, a workshop, three keynotes, a full technical program of refereed papers and industrial presentations, an industrial exhibition and vendor presentations, and a social program.

Eight excellent tutorials on Monday and Friday cover a broad range of topics: Embedded ARM Programming with Ada 2012; Parallelism in Ada, C, Java and C#, Today and Tomorrow; A Semi-formal Approach to Software Development; Software Test and Verification Techniques for Dependable Systems; Ada 2012 (Sub)types and Subprogram Contracts in Practice; Towards Energy Awareness and Predictability in the Linux Kernel; Access Types and Memory Management in Ada 2012; Using Gnoga for Desktop/Mobile GUI and Web development in Ada.

In addition, on Friday the conference hosts for the 3rd consecutive year the International Workshop on "Challenges and new Approaches for Dependable and

Cyber-Physical Systems Engineering" (De-CPS 2016).

Three eminent keynote speakers have been invited to open each day of the core conference program. Alan Burns, in "Why the Expressive Power of Languages such as Ada is needed for Future Cyber Physical Systems", shows how Ada provides programming abstractions to exploit the wealth of real-time scheduling theory available to obtain efficient resource utilization. Valerio Giorgetta, in "Challenges for the Automotive Platform of the Future", shows how cars will be impacted by the various technologies currently in development. Marc Duranton, in "The HiPEAC Vision", presents the roadmap of the European Network on High Performance and Embedded Architecture and Compilation to address the upcoming challenges in computing systems.

The technical program presents 12 refereed and carefully selected papers on the latest research, new tools, applications and industrial practice and experience, a collection of 8 industrial presentations reflecting current practice and challenges, 2 presentations and a discussion in a special "Ada & Parallelism" session, a project presentation and a poster session, and vendor presentations. Springer Verlag publishes keynote talks and all peer-reviewed papers in the proceedings of the conference, as LNCS Vol. 9695. The remainder of the proceedings will be published in the Ada User Journal, the quarterly magazine of Ada-Europe.

The industrial exhibition opens Tuesday morning and runs until the end of Thursday afternoon. Exhibitors include AdaCore, Ansys/Esterel, PTC Developer Tools, Rapita Systems, Vector Software, and Ada-Europe.

The social program includes a Welcome Reception on Tuesday evening in the garden of the Scuola Superiore Sant'Anna, and on Wednesday evening the traditional Ada-Europe Conference Banquet will be held at the cloister of Santa Maria del Carmine in the pedestrian area in the center of Pisa close to the Arno river. Each day, coffee breaks in the exhibition area and sit-down lunches offer ample time for interaction and networking.

The Best Paper Award will be presented during the Conference Banquet, the Best Presentation Award during the Closing session.

The conference is hosted by the Scuola Superiore Sant'Anna, an internationally renowned university school located in the heart of Pisa. The Scuola can be easily reached from the Campo dei Miracoli airport or the railway station.

The full program is available on the conference web site.

Online registration is still possible.

Latest updates:

The "Final Program" is available at

http://www.ada-europe.org/conference2016/AE2016_final_program.pdf

Check out the 8 tutorials in the PDF program, or in the schedule at

http://www.ada-europe.org/conference2016/tutorials.

Registration fees are very reasonable and the registration process is done on-line. Don't delay! For all details, see

http://www.ada-europe.org/conference2016/reg.

The proceedings, published by Springer Verlag as Lecture Notes in Computer Science Vol. 9695, are ready and will be distributed at the conference. See http://www.springer.com/gp/book/9783319390826.

Help promote the conference by advertising for it!

http://www.ada-europe.org/conference2016/promo.

Recommended Twitter hashtags: #AdaEurope and/or #AdaEurope2016.

The 16-page "Advance Program" is still available for viewing at

http://www.ada-europe.org/conference2016/AE2016_advance_program_lq.pdf

and for printing at

http://www.ada-europe.org/conference2016/AE2016_advance_program.pdf

For the latest information consult the conference web site

http://www.ada-europe.org/conference2016.

"Make with Ada" Programming Competition

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Mon, 20 Jun 2016 10:45:19 -0700
Subject: AdaCore Launches "Make with Ada" Programming Competition, with €5000 Top Prize
Newsgroups: comp.lang.ada

AdaCore today announced the launch of the "Make with Ada" programming competition, a contest that aims to help the embedded software community improve the quality of their code by encouraging the use of the Ada and SPARK programming languages. The competition will run from June 20 to September 30, 2016 and offers over €8000 in total prizes. Participants can register for the competition at www.makewithada.org.

Competition Rules

The competition is open to individuals and to teams with up to four members. The goal is to design and implement an embedded software project for an ARM Cortex M or R processor where Ada and/or SPARK are the principal language technologies. Entrants will need to demonstrate that their system meets its requirements and has been developed using sound software engineering practices. Submission deadline is September 30, 2016. The award winners will be announced in November 2016.

Prizes and Judging Criteria

Cash prizes will be awarded to the projects that best meet the overall criteria of software dependability, openness, collaborativeness and inventiveness.

- Top Prize - 5000 Euros

- Second Prize – 2000 Euros

- Third Prize – 1000 Euros

Two special awards (nano-drones) will also be offered: one for the project rated best for dependability, and the other for the project rated best for inventiveness.

Judges

The competition judges include embedded systems experts Jack Ganssle, Principal Consultant at The Ganssle Group; Dick Selwood, Europe Editor at TechfocusMedia; Bill Wong, Technical Editor at Penton Media; and Cyrille Comar, AdaCore President.

"Building an application in Ada on a deeply-embedded microcontroller like the Cortex M or R will be a ton of fun, and is a great way to demonstrate how Ada leads to great code," said competition judge Jack Ganssle.

"This is an exciting opportunity for developers to try a new technology and show their imagination and programming talents," said Fabien Chouteau, AdaCore software engineer and author of the Make With Ada blog post series. "Ada is most known for its usage in large-scale long-lived systems but it is also an excellent tool even for the most humble embedded project."

The "Make with Ada" competition is part of an overall AdaCore initiative to foster the growth of Ada and SPARK for developing embedded systems and more generally for developing "software that matters". Other elements of this initiative are the free on-line training available at AdaCore U, and the various resources for free software developers and students/hobbyists at the github repository and the libre site.

Further information about Ada and SPARK, along with links to free resource pages and instructions on how to get started by downloading the GNAT GPL edition for Bare Board ARM, are available at makewithada.org/getting-started.

Ada-related Resources

Ada on Social Media

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sun Jul 3 2016
Subject: Ada on Social Media

Ada groups on various social media:

- LinkedIn: 2_425 members [1]
- Reddit: 870 readers [2]
- Google+: 668 members [3]
- StackOverflow: 530 followers [4]
- Freenode: 78 participants [5]
- Twitter: 8 tweeters [6]

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

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

[3] https://plus.google.com/communities/102688015980369378804

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

[5] #Ada on irc.freenode.net

[6] https://twitter.com/search?f=realtime&q=%23AdaProgramming

[See also "Ada on Social Media", AUJ 37-1, p. 6. —sparre]

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sun Jul 3 2016
Subject: Repositories of Open Source software

GitHub: 1_277 repositories [1]
332 developers [1]
1_064 issues [1]

Rosetta Code: 626 examples [2]
30 developers [3]
0 issues [4]

Sourceforge: 247 repositories [5]

BlackDuck OpenHUB: 211 projects [6]

Bitbucket: 76 repositories [7]

OpenDO Forge: 24 projects [8]
491 developers [8]

Codelabs: 14 repositories [9]

AdaForge: 8 repositories [10]

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

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

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

[4] http://rosettacode.org/wiki/Category:Ada_examples_needing_attention

[5] http://sourceforge.net/directory/language%3Aada/

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

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

[8] https://forge.open-do.org/

[9] http://git.codelabs.ch/

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

[See also "Repositories of Open Source Software", AUJ 37-1, p. 6. —sparre]

Ada-related Tools

Reading Microsoft Excel Files

From: Jean François Martinez <darkquark99@gmail.com>
Date: Tue, 1 Mar 2016 02:42:38 -0800
Subject: Reading Excel 2010 files with Ada on Linux
Newsgroups: comp.lang.ada

Anyone knows if there is an Ada package for reading xlsx files? The program will run on Linux so the package must not depend on Microsoft libraries.

I have googled around but all what I have found is a package for writing Excel files not for reading them.

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Tue, 1 Mar 2016 06:42:54 -0800
Subject: Re: Reading Excel 2010 files with Ada on Linux
Newsgroups: comp.lang.ada

An xlsx file is actually a Zip file containing XML files.

The first step would be to open the file or stream with Zip-Ada [1], then parse the appropriate XML entry with XML/Ada [2].

[1] http://unzip-ada.sf.net

[2] http://libre.adacore.com/tools/xmlada/

Simple Components

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 1 Mar 2016 22:48:02 +0100
Subject: ANN: Simple Components for Ada v4.11 released
Newsgroups: comp.lang.ada

The library 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, multiple connections server/client designing tools.

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

Changes to the previous version:

- ELV/e-Q3 MAX! client protocol implementation corrected;
- ELV/e-Q3 MAX! client supports reading measured temperature from radiator thermostats;
- ELV/e-Q3 MAX! client subprograms Get_Error, Has_Device_Data, Query_NTP_Servers, Reset_Devices, Reset_Error, Set_NTP_Servers added;
- ELV/e-Q3 MAX! client messages and commands A, F, N support added;
- GNUTLS bindings updated to the latest version.

[See also "Simple Components", AUJ 36-4, p. 202. —sparre]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 12 Apr 2016 20:48:03 +0200
Subject: Simple Components 4.12 with MQTT implementation released
Newsgroups: comp.lang.ada

[...]

This new version provides a full implementation of MQTT protocol. The implementation includes a raw MQTT stack for custom clients/servers and a full implementation of a broker with persistent sessions support, retained topics and bulk messages publishing. The broker implementation also provides some enhancements with regard of topic publishing in order to alleviate the MQTT protocol drawbacks. Stream interfaces to the message content are supported as well.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 30 May 2016 22:36:59 +0200
Subject: ANN: Simple Components v4.13 released
Newsgroups: comp.lang.ada

[...]

This version fixes bugs in the MQTT broker.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 20 Jun 2016 19:14:40 +0200
Subject: ANN: Simple Components for Ada v4.14 with SMTP client
Newsgroups: comp.lang.ada

The new version provides an implementation of SMTP client. As other protocols implementations provided by the library, this one is driven by the multiple-connections server so that a single Ada task can handle multiple connections and multiple protocols.

The implementation is asynchronous capable to send more than one mail. A simplified synchronous (blocking) variant

is provided as well. MIME/attachments are supported. SSL/TLS is also possible in both its variants: sessions encrypted from the start and opportunistic TLS sessions AKA STARTTLS. The TLS support is based on GNUTLS.

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

Bug reports and feature requests are welcome.

GtkAda Contributions

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 3 Mar 2016 18:09:27 +0100
Subject: ANN: GtkAda contributions v3.15 released
Newsgroups: comp.lang.ada

The library extends GtkAda providing:

- Tasking support;
- Custom models for tree view widget;
- Custom cell renderers for tree view widget;
- Multi-columned derived model;
- Extension derived model (to add columns to an existing model);
- Abstract caching model for directory-like data;
- Tree view and list view widgets for navigational browsing of abstract caching models;
- File system navigation widgets with wildcard filtering;
- Resource styles;
- Capturing resources of a widget;
- Embeddable images;
- Some missing subprograms and bug fixes;
- Measurement unit selection widget and dialogs;
- Improved hue-luminance-saturation color model;
- Simplified image buttons and buttons customizable by style properties;
- Controlled Ada types for GTK+ strong and weak references;
- Simplified means to create lists of strings;
- Spawning processes synchronously and asynchronously with pipes;
- Capturing asynchronous process standard I/O by Ada tasks and by text buffers;
- Source view widget support.

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

Changes to the previous version:

- Get_Clip_Rectangle procedure and function added to Gtk.Missed;
- Freeze_Notify and Thaw_Notify added to Gtk.Missed.

[See also “GtkAda Contributions”, AUJ 36-4, p. 202. —sparre]

Industrial Control Widget Library

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 4 Mar 2016 17:29:06 +0100
Subject: ANN: Ada Industrial Control Widget Library v3.13 released
Newsgroups: comp.lang.ada

The software is based on GtkAda, Ada bindings to GTK+ and Cairo. The key features of the library:

- Widgets composed of transparent layers drawn by cairo;
- Fully scalable graphics;
- Support of time controlled refresh policy for real-time and heavy-duty applications;
- Caching graphical operations;
- Stream I/O support for serialization and deserialization;
- Ready-to-use gauge, meter, oscilloscope widgets;
- Editor widget for WYSIWYG design of complex dashboards.

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

Changes to the previous version:

- Minor changes improving performance;
- Waveform sweeper interface is enhanced to suppress the "draw" signal flood when a sweeper is shared by several independent widgets.

[See also “Industrial Control Widget Library”, AUJ 36-4, p. 202. —sparre]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 13 Apr 2016 18:33:47 +0200
Subject: Ada industrial control widget library 3.14 released
Newsgroups: comp.lang.ada

The library is provided for design high-quality industrial control widgets for Ada applications. The software is based on GtkAda, Ada bindings to GTK+ and cairo.

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

Changes to the previous version:

- In Gtk_Oscilloscope, when the selection mode is set to None, the selection highlighting on left button press is turned off;
- The right button click drop-down Gtk_Oscilloscope menu item Latest data is shown only if at least one sweeper does render time;
- Gtk_Oscilloscope drop-down menu items can be individually enabled and disabled;
- Add_Group of Gtk_Oscilloscope accepts an amplifier object to use with the group;
- Set_Renderer was added to Gtk.Layered.Graph_Paper_Annotation;
- Get_Suffix and Set_Suffix were added to Gtk.Layered.Graph_Paper_Annotation;
- Get_Time_Tooltip_Suffix, Get_Tooltip_Annotation, Get_Values_Tooltip_Suffix, Set_Time_Tooltip_Suffix, Set_Tooltip_Annotation, Set_Values_Tooltip_Suffix were added to Gtk_Oscilloscope;
- Function Image was added to Gtk.Layered.Graph_Paper_Annotation;
- Extrapolation left and right was added to Gtk.Layered.Waveform and Gtk.Oscilloscope;
- Get_Release_To_Latest and Set_Release_To_Latest were added to Gtk.Oscilloscope;
- Undo/redo stubs with pattern matched names added to Gtk.Oscilloscope.

ZLib-Ada

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Tue, 8 Mar 2016 13:35:43 -0800
Subject: Poll for the users of ZLib-Ada
Newsgroups: comp.lang.ada

What is the lowest compression level that you need usually ?

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Sat, 12 Mar 2016 21:37:45 +0100
Subject: Re: Poll for the users of ZLib-Ada
Newsgroups: comp.lang.ada

> [...]

I think we use whatever the default is.

We use it to zip logfiles after they grow too large.

I guess we are happy with the level of compression we get, I don't think there has ever been a question raised addressing the level.

Pretty-printing Tools

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sun, 13 Mar 2016 19:07:32 +0100
Subject: Pretty-printer? (Alternatives to "gnatpp")
Newsgroups: comp.lang.ada

What pretty-printers are available for Ada (83-2012)?

I'm already aware of "gnatpp", but it has a disadvantage:

1) You have to have set up the environment to compile the source file, before you can pretty-print it.

What alternatives are there?

How do they fare compared to "gnatpp"?

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Sun, 13 Mar 2016 22:42:07 +0100
Subject: Re: Pretty-printer? (Alternatives to "gnatpp")
Newsgroups: comp.lang.ada

The (new) Ada mode for Emacs can be used as a pretty printer, for all versions of Ada, and it supports automatic letter case. The user may define a list of exceptions to the casing rules that he prefers.

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Sun, 13 Mar 2016 17:58:13 -0700
Subject: Pretty-printer? (Alternatives to "gnatpp")
Newsgroups: comp.lang.ada

Byron is at the lexing stage, it could therefore be used as input to a pretty-printer... or just add a "filter" function that has a local dictionary w/ identifiers of the casing you want and iterate over the token-array with the following conditional:

IF token is identifier THEN

IF identifier not present THEN add identifier to dictionary

OTHERWISE replace identifier w/ one of the found casing.

END IF

END IF

https://github.com/OneWingedShark/Byron

From: Britt <britt.snodgrass@gmail.com>
Date: Mon, 14 Mar 2016 17:15:44 -0700
Subject: Re: Pretty-printer? (Alternatives to "gnatpp")
Newsgroups: comp.lang.ada

For formatting single files as you edit, I've been happy enough with the formatting capabilities build into GPS and GNATbench. These don't require files to be completely compilable.

Cortex GNAT Run Time Systems

From: Simon Wright <simon@pushface.org>
Date: Mon, 14 Mar 2016 17:42:22 +0000
Subject: ANN: Cortex GNAT RTS 20160314
Newsgroups: comp.lang.ada

At https://sourceforge.net/projects/cortex-gnat-rts/files/20160314/.

This release includes

- an RTS for the Arduino Due, arduino-due, and a minimal BSP, arduino-due-bsp.
- an RTS for the STM32F429I-DISCO, stm32f429i-disco-rtos, based on STMicroelectronics' STM32Cube package and FreeRTOS, and a corresponding partial BSP, stm32f429i-disco-bsp.
- an RTS for the STM32F429I-DISCO, stm32f429i, based on FreeRTOS, with a set of peripheral definition packages created by SVD2Ada.

In this release,

- the Containers support generalized iteration ("for all E of C loop"). Note, this is achieved by removing tampering checks. While tampering errors are rare, it would be as well to check algorithms using a fully-featured desktop compiler.

The standard packages included (there are more, implementation-specific, ones) are:

- Ada
- Ada.Containers
- Ada.Containers.Bounded_Hashed_Maps
- Ada.Containers.Bounded_Vectors
- Ada.Exceptions
- Ada.IO_Exceptions
- Ada.Interrupts
- Ada.Interrupts.Names
- Ada.Iterator_Interfaces
- Ada.Real_Time
- Ada.Streams
- Ada.Synchronous_Task_Control
- Ada.Tags
- Ada.Task_Identification
- Interfaces
- Interfaces.C
- Interfaces.C.Strings
- System
- System.Assertions
- System.Address_To_Access_Conversions
- System.Storage_Elements

[See also “Cortex GNAT Run Time Systems”, AUJ 37-1, p. 15. —sparre]

From: Simon Wright <simon@pushface.org>
Date: Sun, 22 May 2016 15:20:39 +0100
Subject: ANN: Cortex GNAT RTS 20160522
Newsgroups: comp.lang.ada

Available at

https://sourceforge.net/projects/cortex-gnat-rts/files/20160522/

This release includes GNAT Ada Run Time Systems (RTSs) based on FreeRTOS (http://www.freertos.org) and targeted at boards with Cortex-M3, -M4, -M4F MCUs (Arduino Due from http://www.arduino.org, the STM32F4-series evaluation boards from STMicroelectronics at http://www.st.com).

In each case, the board support for the RTS (configuration for size and location of Flash, RAM; clock initialization; interrupt naming) is in $RTS/adainclude. Support for the on-chip peripherals is also included, in Ada spec files generated by SVD2Ada (https://github.com/AdaCore/svd2ada).

The Ada source is either original or based on FSF GCC (mainly 4.9.1, some later releases too).

(1) arduino-due is a Ravenscar-style RTOS based on FreeRTOS from http://www.freertos.org for the Arduino Due.
See arduino-due/COPYING* for licensing terms.
On-chip peripheral support in atsam3x8e/.
Tests in test-arduino-due/.

(2) stm32f4 is a Ravenscar-style RTOS based on FreeRTOS from http://www.freertos.org for the STM32F4-DISC* board.
See stm32f4/COPYING* for licensing terms.
On-chip peripheral support in stm32f40x/.
Tests in test-stm32f4/.

(3) stm32f429i is a Ravenscar-style RTOS based on FreeRTOS from http://www.freertos.org for the STM32F429I-DISC* board.
See stm32f429i/COPYING* for licensing terms.
On-chip peripheral support in stm32f429x/.
Tests in test-stm32f429i/.

In this release,

- There is no longer any dependence on the STMicroelectronics' STM32Cube package.
- The support for on-chip peripherals is limited to the SVD2Ada-generated spec files. The AdaCore 'bareboard' software (currently https://github.com/AdaCore/bareboard, but a name change is under consideration) supports the STM32 line.
- Tasking no longer requires an explicit start (https://sourceforge.net/p/cortex-gnat-rts/tickets/5/).
- Locking in interrupt-handling protected objects no longer inhibits all interrupts, only those of equal or lower priority (https://sourceforge.net/p/cortex-gnat-rts/tickets/18/).

[...]

MQTT

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 24 Mar 2016 20:15:21 +0100
Subject: MQTT native Ada implementation
Newsgroups: comp.lang.ada

The incoming version of Simple Components will provide a native (not bindings) implementation of the MQTT 3.1.1 stack and on top of it a full MQTT messages broker.

In order to test the implementation I would be thankful for MQTT test use cases.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 26 Mar 2016 10:05:59 +0100
Subject: Re: MQTT native Ada implementation
Newsgroups: comp.lang.ada

> Do you implement support for the TLS transport as well?

Yes. The implementation is driven by a connections server object. The connections server may run several protocols simultaneously. The server itself has a number of implementations, one of them is a secure server that uses GNUTLS as the SSL/TLS layer. That should do it, I think.

MQTT with TLS must be quite slow with the QoS level 2, which deploys a lot of small packets sent here and there. And in general, MQTT design is not very exciting, but it was easy to implement, so I gave it a try.

[See also “Mosquitto”, AUJ 36-3, p. 122. —sparre]

Zip-Ada

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Wed, 30 Mar 2016 23:30:12 -0700
Subject: Ann: Zip-Ada v.50
Newsgroups: comp.lang.ada

There is a new version of Zip-Ada @ http://unzip-ada.sf.net .

In a nutshell, there are now, finally, fast *and* efficient compression methods available.

* Changes in '50', 31-Mar-2016:

- Zip.Compress.Shrink is slightly faster

- Zip.Compress.Deflate has new compression features:

- Deflate_Fixed is much faster, with slightly better compression

- Deflate_1 was added: strength similar to zlib, level 6

- Deflate_2 was added: strength similar to zlib, level 9

- Deflate_3 was added: strength similar to 7-Zip, method=deflate, level 5

As you perhaps know, the Deflate format performs compression in two steps by combining a LZ77 algorithm with Huffman encoding.

In this edition, two known algorithms are combined probably for the first time within the same software.

Additionally, the determination of compressed blocks' boundaries is done by an original algorithm based on similarities between Huffman code sets.

Zip-Ada is a library for dealing with the Zip compressed archive file format. It supplies:

- compression with the following sub-formats ("methods"): Store, Reduce, Shrink (LZW) and Deflate

- decompression for the following sub-formats ("methods"): Store, Reduce, Shrink (LZW), Implode, Deflate, BZip2 and LZMA

- encryption and decryption (portable Zip 2.0 encryption scheme)

- unconditional portability - within limits of compiler's provided integer types and target architecture capacity

- input (archive to decompress or data to compress) can be any data stream

- output (archive to build or data to extract) can be any data stream

- types Zip_info and Zip_Create_info to handle archives quickly and easily

- cross format compatibility with the most various tools and file formats based on the Zip format: 7-zip, Info-Zip's Zip, WinZip, PKZip, Java's JARs, OpenDocument files, MS Office 2007+, Nokia themes, and many others

- task safety: this library can be used ad libitum in parallel processing

- endian-neutral I/O

[See also “Zip-Ada”, AUJ 36-2, p. 63. —sparre]

Thin XCB Binding

From: Joakim Strandberg <joakimds@kth.se>
Date: Sat, 16 Apr 2016 06:56:45 -0700
Subject: Announcing Ada binding to the XCB library
Newsgroups: comp.lang.ada

This may be a bit premature since there are a few functions from "the core library" missing, but I would like to announce the existence of an Ada binding to the XCB library:

https://github.com/joakim-strandberg/xcb_library_thin_ada_binding

Much of the code in the Ada binding is auto-generated from a file called xproto.xml. I have not (yet) uploaded the application that does the parsing and generates the Ada code, but I would like to share with you that I did not base it upon XMLAda, but wrote the XML parsing code by hand (for fun!) and to do the parsing of UTF8 characters I used the code from Strings_Edit from Simple Components written by Dmitry Kazakov. And so, special thanks to him! (I am going to update the README-file with this information too).

Dequesterity

From: Brad Moore <bmoore.ada@gmail.com>
Date: Thu, 21 Apr 2016 23:17:46 -0700
Subject: ANN: Dequesterity v1.5 (Buffers of many shapes and sizes)
Newsgroups: comp.lang.ada

I am pleased to announce the availability of Dequesterity, version 1.5

Dequesterity is a set of Ada 2005 generics that provide various forms of general purpose buffer containers. Buffers may be used as deques, queues, ring buffers, stacks, double ended stacks, vectors, priority queues, and similar abstractions.

There are various concurrent buffers, priority buffers, streaming buffers, remote buffers. In fact there are now over 100 buffer packages to choose from.

This release mostly fixes some issues with the streaming Ravenscar buffers.

Those examples were no longer compiling with the GNAT GPL 2015 version of the compiler. The compilations were failing due to violations of the Ravenscar restrictions, No_Protected_Type_Allocators and No_Local_Protected_Objects.

I believe these are valid restrictions that just were not being caught in earlier versions of the compiler. The problem is that the Stream Buffer has an internal record component that is an internal buffer implemented as a protected type.

To address these restrictions, I had to pull out the internal buffer declaration, and instead have the programmer declare this separately, then on the Stream Buffer declaration, have the programmer reference the "internal" buffer via an access discriminant.

This release and older releases may be downloaded from;

https://sourceforge.net/projects/dequesterity/files/

Thanks to Daniel for alerting me to the fact that the Ravenscar examples were no longer compiling.

[See also “Dequesterity”, AUJ 33-4, p. 237. —sparre]

Excel Writer

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Mon, 25 Apr 2016 06:04:42 -0700
Subject: Ann: Excel Writer v.15
Newsgroups: comp.lang.ada

A quick note to announce the latest release of Excel Writer.

http://excel-writer.sourceforge.net/

The changes since last note here are:

15: 23-Apr-2016:
- zoom factor for viewing / editing sheet

- international code pages were added: Thai, Japanese Shift-JIS, Chinese Simplified GBK, Korean (Wansung), Chinese Traditional BIG5, Latin II (Central European), Cyrillic, Latin I, Greek, Turkish, Hebrew, Arabic, Baltic, Vietnamese, Korean (Johab).

14: 20-Jul-2014:

- cells "locked" (Excel's default, this allows formula protection)
- Next, Next_Row admit zero as parameter

[See also " Excel Writer", AUJ 35-2, p. 80. —sparre]

PDF_Out

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Mon, 25 Apr 2016 06:11:51 -0700
Subject: Ann: Ada PDF Writer v.002
Newsgroups: comp.lang.ada

Here is a note about the latest release of Ada PDF Writer.

http://apdf.sourceforge.net/

Latest changes are:

- ISO Latin-1 character support
- Image dimensions can be queried (useful before inserting them!)
- Added tool: img2pdf
- Improved demo - vector graphics with the Ada Mascot!
- Some fixes

[See also " PDF_Out", AUJ 37-1, p. 7. —sparre]

PragmARC.Unbounded_Integers

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Fri, 29 Apr 2016 10:01:18 -0700
Subject: Modified Version of Unbounded_Integers
Newsgroups: comp.lang.ada

Recent versions of GNAT, including GPL 2015 and gcc 6, have an error that prevents them from compiling PragmARC.Unbounded_Integers. There is no telling when this error might be corrected.

Also, some people have reported confusion with having two procedures with the same simple name and the same parameter profile.

To accommodate both groups, the name of one of these procedures has been changed in the current version of the beta PragmARCs. As this change is confined to the package body, it will have no effect on those who have used an earlier version of the package.

The new version may be obtained from the PragmAda web site or from the Github repository.

[See also " PragmAda Reusable Components", AUJ 37-1, p. 14. —sparre]

Imago

From: Tomasz "darkestkhan" Maluszycki <darkestkhan@gmail.com>
Date: Sun, 1 May 2016 12:12:55 -0700
Subject: ANN: Imago 0.2 released.
Newsgroups: comp.lang.ada

https://github.com/darkestkhan/imago/tree/0.2

Imago is a thin binding to DevIL - Developers' Image Library (which is a library that supports working with most image formats). It closely follows original API so for how information on how to use it I recommend looking at original library documentation and lazyfoo tutorials (and just happens so that I have repository on github with lazyfoo tutorials implemented in Ada: www.github.com/darkestkhan/lazyfoo)

AdaBase

From: John Marino <dragonlace.cla@marino.st>
Date: Fri, 13 May 2016 13:37:15 -0700
Subject: ANN: Introducing AdaBase - Thick database bindings for Ada
Newsgroups: comp.lang.ada

Hey guys, I know there are several options for Ada to interface with databases, but I wasn't happy with any of them for various reasons and thus created yet another option to scratch my itch. I was aiming at a consistent interface to which the various drivers adapt. With some care, the database backends should be interchangeable.

So far I've created drivers for MySQL and SQLite and the driver for PostgreSQL is next on my list. I'd like to eventually support others such as Firebird, MSSQL, Oracle, etc., but those will be very low priority for me over the next year. Contributions are welcome of course -- It's been released under the developer- and commercial-friendly ISC license.

I've spent a lot of time documenting the interface and providing a lot of real examples. If you are at all looking for something like this, I recommend that you spend a few minutes going through the descriptions and examples of all the functions:

http://jrmarino.github.io/AdaBase/

It's been developed on DragonFly and FreeBSD, and I haven't tested it on Windows yet, but I will. it's already available for BSD users, see: http://www.freshports.org/databases/adabase

I believe AdaBase is already mature for MySQL and SQLite, but reports of issues will be welcome. I'm active on github, so issues and pull requests will be dispositioned quickly if you wish to leverage those tools.

Hopefully other people will find this project useful!

From: John Marino <dragonlace.cla@marino.st>
Date: Thu, 16 Jun 2016 13:33:05 -0700
Subject: ANN: AdaBase 3.1 - includes Spatial data / Geographic object support
Newsgroups: comp.lang.ada

I had intended that release 3.1 of AdaBase be a quick improvement that would add three new native data types:

1) Bit type (like bit flags)
2) UTF8 encoded strings
3) Spatial data types / OpenGIS Geometry

The first two were relatively straight-forward. Bits were sort of already supported but inconsistently and sometimes as strings. Now they are an array of bits for easy manipulation.

AdaBase tries to be smart with regard to encoding. It forces the server to send text strings encoded as UTF8 and stores it natively and decodes them to strings/w/ws/ as necessary.

The support for MySQL spatial data types and the PostGIS extension for PostgreSQL took far longer than I anticipated. The result is pretty good, I think, it allows AdaBase to directly query geometry fields without the use of database server-side functions to convert and extract. It reads the internal format of MySQL and PostGIS directly and converts them to Well Known Binary (WKB) which can be used to construct AdaBase Geometry objects and also into Well Known Text. This eliminates a lot of overhead as point data can be directly used immediately after a simple query that pulls raw column data.

I wrote a pretty log documentation[1] page with two separate test cases to show geometry data extraction and the geometry data construction and insertion. I don't know if anybody needs this feature, but I have a feeling there aren't a lot of database API packages that can handle GIS data like this.

[1] https://jrmarino.github.io/AdaBase/geometry.html

[2] https://github.com/jrmarino/AdaBase

Stream Tools

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Sat, 21 May 2016 13:38:25 +0200
Subject: [ANN] stream-tools 1.1.0
Newsgroups: comp.lang.ada

The suite now contains:

- A Stream implementing in-memory pipes.
- Debug stream readers that prints to standard output.

https://github.com/persan/a-stream-tools/releases/tag/1.1.0-2016-05-19

[See also " Stream Tools", AUJ 37-1, p. 16. —sparre]

SymExpr

From: Riccardo Bernardini <framefritti@gmail.com>
Date: Sun, 22 May 2016 08:46:46 -0700
Subject: ANN: SymExpr 1.1 (parsing and manipulating expressions)
Newsgroups: comp.lang.ada

it seems that I never announced this package that I wrote for the usual "itch of mine." It is a small package that parses, manipulates and evaluates basic expressions (four operations, unary + and -, function calls and variables). Maybe later I'll extended it to be more general.

I wrote this piece of code because I needed it for another program of mine. After finishing writing the code, I discovered that with minimal effort I could have made it a stand-alone package, and SymExpr was born.

If you want to know more, check out the README of version 1.1 at

https://launchpadlibrarian.net/134442100/README

If you want to try it,

https://launchpad.net/symexpr/+download

As with all this kind of projects, I wrote it because I needed it; if it is useful to you, you are welcome to use it. If you have suggestions and/or requests, I am open to them, although I cannot guarantee that I will be able to satisfy them.

If you use it somewhere, it would improve my self-esteem to know it :-) :-) :-)

Adequate

From: Rolf Ebert <rrr.eee.27@gmail.com>
Date: Tue, 31 May 2016 00:03:39 -0700
Subject: ANN: Adequate, MQTT broker and client programs based on Kasakov's Components
Newsgroups: comp.lang.ada

The Adequate project provides command line clients for the MQTT protocol. All the hard work is provided by Dmitry Kazakov's Components. Among them is an implementation of the MQTT protocol.

The Adequate project consists of three programs

1. aq_broker, a MQTT broker or MQTT server.
2. aq_pub, a MQTT client for publishing single messages with a command line interface.
3. aq_sub, a MQTT client for subscribing to a topic (including wildcards). It also only has a command line interface.

You can find the Github project here:

https://github.com/RREE/adequate

All programs are still in their infancy. They should help you getting started with Dmitry's code.

If you want to direct the future development of Adequate I encourage you to create Issues (https://github.com/RREE/adequate/issues). I also welcome adding and extending the wiki pages (https://github.com/RREE/adequate/wiki) or Pull Requests.

GNAT GPL and SPARK GPL

From: Nasser M. Abbasi <nma@12000.org>
Date: Wed, 1 Jun 2016 08:33:56 -0500
Subject: fyi, GNAT and SPARK GPL 2016 are out
Newsgroups: comp.lang.ada

got this email from Ada core today:

Dear GNAT and SPARK GPL user,

We are pleased to announce the availability of the GNAT and SPARK GPL 2016 toolsets.

GNAT GPL 2016 incorporates upgraded technology for the debugger (GDB 7.10) along with support for the Windows 10 platform and many new features.

Ada runtime support has been extended for the STM32f429-disco, STM32f469-disco and STM32F7-disco development boards based on the STM32 family of microcontrollers.

- ravenscar sfp/full for the stm32f429-disco board
- ravenscar sfp/full for the stm32f469-disco board
- ravenscar sfp/full for the stm32f7-disco board

SPARK GPL 2016 - the formal method verification toolset - includes the following new features:

- Support for concurrency with Ravenscar and type predicates
- Generation of counterexamples for unproved checks
- Better support of bitwise (modular) operations in proof
- Generation of global summary table

You will find documentation about the GNAT GPL 2016 and SPARK GPL 2016 toolset here:

http://libre.adacore.com/developers/documentation

Both toolsets can be downloaded:

- from the "Download" section on GNAT Tracker for GAP users http://www.adacore.com/academia
- from libre site libre.adacore.com

GNATColl.JSON Support Packages

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Thu, 2 Jun 2016 10:25:06 +0200
Subject: [ANN] gnatcoll-JSON-v0.0.6
Newsgroups: comp.lang.ada

Provides a set of support packages to for JSON serialization/deserialization of to Ada.Containers and some other types in the "standard" packages.

This is the first release:

https://github.com/persan/gnatcoll-json/releases/tag/gnatcoll-JSON-v0.0.6

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Tue, 14 Jun 2016 20:35:43 +0200
Subject: [ANN] gnatcoll-JSON-v1.0.1
Newsgroups: comp.lang.ada

[...]

https://github.com/persan/gnatcoll-json/releases/tag/gnatcoll-JSON-v1.0.1

PragmARC.Text_IO

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Fri, 3 Jun 2016 09:28:37 -0700
Subject: New Version of PragmARC.Text_IO
Newsgroups: comp.lang.ada

There's a new version of package PragmARC.Text_IO available. This corrects an error in Skip_Line and changes the line terminator used for output from a single value used for all files to a value per file, specified when the file is opened or created.

The PragmAda Reusable Components are available from the web site or from

https://github.com/jrcarter/PragmARC

[See also "PragmARC.Text_IO", AUJ 37-1, p. 15. —sparre]

I2C and SPI Support for STM32F411RE Nucleo Board

From: Sean Day <seanjdayatd@gmail.com>
Date: Sun, 12 Jun 2016 23:48:26 -0700
Subject: STM32F411RE Nucleo Board
Newsgroups: comp.lang.ada

After a long break from Ada I have attempted to get GNAT for bare-board ARM running on an STM32F411RE Nucleo board.

I successfully got the Led_Demo running and after some further investigation got GNAT.IO working.

However, I have been unable to find working examples of I2C or SPI sensors.

Any useful links?

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Mon, 13 Jun 2016 02:00:53 -0700
Subject: Re: STM32F411RE Nucleo Board
Newsgroups: comp.lang.ada

Have a look here: [1]. It's a library of bare-metal drivers for Ada with an HAL that allows portable sensor drivers over I2C, SPI, UART, etc.

Contributions are welcome :)

[1] https://github.com/AdaCore/Ada_Drivers_Library

From: Simon Wright <simon@pushface.org>
Date: Mon, 13 Jun 2016 10:07:46 +0100
Subject: Re: STM32F411RE Nucleo Board
Newsgroups: comp.lang.ada

I did a little bit of unfinished work for I2C here:

http://cloud.likeabird-group.eu:7990/scm/~sjw/multiplexed-io.git

(subdirectory pcf8574a).

This was using SVD2Ada (https://github.com/AdaCore/svd2ada).

GLOBE_3D

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Wed, 15 Jun 2016 04:22:19 -0700
Subject: Ann: GLOBE_3D Release 2016-06-12 - Added multitexturing
Newsgroups: comp.lang.ada

A bit of progress - after a few years of hibernation...

The news here are: multitexturing (so far, diffuse + specular).

Surfaces reflect now light in a more realistic way, provided specific textures for specular reflection are provided.

GLOBE_3D is a GL Object Based 3D engine realized with the Ada programming language.

URL: http://globe3d.sf.net

First video captures published:

https://www.youtube.com/watch?v=Bf7kyxVVIXs

https://www.youtube.com/watch?v=_lEWKx2lZ88

PUGIXML_Ada

From: Warren Gay <ve3wwg@gmail.com>
Date: Tue, 14 Jun 2016 21:19:42 -0700
Subject: Announce: pugixml_ada -- pugixml Thick Binding for Ada
Newsgroups: comp.lang.ada

This is a quick announcement of a new thick Ada binding for pugixml. If you like the simplicity and ease of use of pugi::xml in C++, you may want to use pugixml_ada in your Ada projects.

The github README shows a simple of a XML config load. Demo program pugidemo.adb also demonstrates the creation of an XML config file.

See

https://github.com/ve3wwg/pugixml_ada

The pugixml project is hosted here:

http://pugixml.org/

XML Output

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 24 Jun 2016 07:44:54 -0700
Subject: generating an XML file using Ada code
Newsgroups: comp.lang.ada

[...] I need to log all db operations to either xml or json.

Using XMLAda, I don't see any way to output a file; I searched for "Text_IO" in the installed library code, and found only input functions. One of the xml tutorials mentions the C function "xmlSaveFormatFile"; I can't find that in the Ada code.

This question has been asked before on this list, several years ago; I hope there is a better answer now.

How can I create an XML file?

For JSON, there is the GNATColl.JSON package, which provides a "Write" function that serializes a JSON object to a string. So I can use that, but I prefer XML (it feels more Ada-like :).

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sat, 25 Jun 2016 08:32:57 +0300
Subject: Re: generating an XML file using Ada code
Newsgroups: comp.lang.ada

> [...]

I have used XML EZ Out, http://www.mckae.com/xmlEz.html, with good results.

From: Simon Wright <simon@pushface.org>
Date: Sat, 25 Jun 2016 08:11:50 +0100
Subject: Re: generating an XML file using Ada code
Newsgroups: comp.lang.ada

> How can I create an XML file?

In ASIS2XML[1] I say

```
DOM.Core.Nodes.Print (Doc,
                      Print_Comments => True,
                      Print_XML_PI => True,
                      EOL_Sequence => "");
```

This produces a single very long line of XML; tidy will make a readable version.

Looking at DOM.Core.Nodes.Print,

```
procedure Print
  (N : Node;
   Print_Comments : Boolean := False;
   Print_XML_PI : Boolean := False;
   With_URI  : Boolean := False;
   EOL_Sequence : String  :=
          Sax.Encodings.Lf_Sequence;
   Encoding  : Unicode.Encodings.
          Unicode_Encoding :=
    Unicode.Encodings.Get_By_Name
          ("utf-8");
   Collapse_Empty_Nodes : Boolean :=
          False);
-- For debugging purposes only!
-- Same as Write, but the output is done on
-- Stdout.
-- Warning: the default values for the
-- parameters are not the same as for
-- write. For the latter, they are chosen so
-- that by default the output is valid XML,
-- whereas Print is mostly intended to be
-- used for testsuite purposes, and the <
-- default match that goal.
```

I see I should have used DOM.Core.Nodes.Write! Another example of scratch development code making it into production.

[1] https://sourceforge.net/p/asis2xml/code/ci/default/tree/asis2xml.adb

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Sat, 25 Jun 2016 16:14:31 +0200
Subject: Re: generating an XML file using Ada code
Newsgroups: comp.lang.ada

> Using xmlada, I don't see any way to output a file;

Maybe print a DOM like this?

http://docs.adacore.com/xmlada-docs/dom.html#printing-dom-tress

From: Vadim Godunko <vgodunko@gmail.com>
Date: Sun, 26 Jun 2016 07:15:14 -0700
Subject: Re: generating an XML file using Ada code
Newsgroups: comp.lang.ada

> How can I create an XML file?

You can use Matreshka's XML.SAX.Pretty_Writers package to generate XML text file and format it (if necessary). It uses "SAX events" to serialize data, thus you don't need to construct whole XML DOM tree in memory before writing to disk.

PragmAda Reusable Components

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Mon, 27 Jun 2016 15:36:53 -0700
Subject: New Version of PragmARCs for ISO/IEC 8652:2007
Newsgroups: comp.lang.ada

There is a new version of the PragmAda Reusable Components for compilers that implement ISO/IEC 8652:2007 available. It introduces a component for job pools (PragmARC.Job_Pools) and contains some improvements to PragmARC.B_Strings.

The PragmARCS are available from the web site or from the repository at

https://github.com/jrcarter/PragmARC

[See also " PragmAda Reusable Components", AUJ 37-1, p. 14. —sparre]

Ada-related Products

Rapita Verification Suite

From: Rapita Systems
Date: Mon Jun 27 2016
Subject: Rapita launches RVS 3.5
URL: https://www.rapitasystems.com/news/rapita-launches-rvs-35

Rapita Systems is proud to announce the latest release of its on-target software verification tool suite RVS, version 3.5.

Our team have worked tirelessly to make this the highest quality release yet, with a huge number of improvements and fixes since 3.4.

We work with customers who are developing to the highest standards of safety and mission critical software. For this reason, the quality of our tools is paramount. With this release, we have raised the bar higher still.

A quick summary of the improvements in this version:

- New MC/DC optimization options
- Support for newer versions of GNAT Pro
- Support for a number of new Ada language features
- Qualification support for code masking
- Improved support for 16-bit DSP chips
- Many improvements to support qualified use of options that reduce instrumentation overheads
- Compiler wrappers can now be used in qualified integrations
- Hundreds of minor bug fixes

Get in touch with us at enquiries@rapitasystems.com for more information on RVS. Registered users can download the latest version through our downloads manager.

What is RVS?

RVS (Rapita Verification Suite) measures, optimizes and verifies the timing performance and test effectiveness of critical real-time embedded systems in industries such as aerospace and automotive. RVS enables aerospace and automotive electronics engineers to tackle the challenges of developing and maintaining critical real-time embedded systems. Qualification support is available for both RapiTime and RapiCover within projects requiring DO-178B/C or ISO 26262 certification.

RVS includes the following plugins:

- RapiTime™ – Performance measurement and timing analysis, including worst-case execution time (WCET)
- RapiCover™ – Structural coverage analysis, including MC/DC coverage
- RapiTask™ – Visualization of RTOS scheduling and event tracing

[See also " Rapita Verification Suite", AUJ 36-4, p. 204. —sparre]

Ada and Operating Systems

FreeBSD: PragmAda Reusable Components

From: John Marino <dragonlace.cla@marino.st>
Date: Fri, 4 Mar 2016 07:05:44 -0800
Subject: Re: New Version of PragmARC.Text_IO
Newsgroups: comp.lang.ada

Thanks for moving this to Github. It made it easy for me to add PragmARCs to FreeBSD ports:

http://www.freshports.org/devel/pragmarcs/

[...]

FreeBSD: Simple Components

From: John Marino <dragonlace.cla@marino.st>
Date: Fri, 4 Mar 2016 05:50:02 -0800
Subject: Re: ANN: Simple Components for Ada v4.11 released
Newsgroups: comp.lang.ada

I updated the FreeBSD port to the latest version:

http://www.freshports.org/devel/simple_components/

Mac OS X: GCC

From: Simon Wright <simon@pushface.org>
Date: Sat, 07 May 2016 14:58:02 +0100
Subject: ANN: GCC 6.1.0 for OS X El Capitan
Newsgroups: comp.lang.ada

This compiler is released in both native and cross-compiler configurations at:

https://sourceforge.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/6.1.0/

Compilers included: Ada, C, C++, Objective C, Objective C++, Fortran.

Tools included:

Full GPL:

ASIS, AUnit, GDB, and GNATColl from GNAT GPL 2015.

GPL with Runtime Library Exception[1]:

- Gprbuild from the public Git repository[2] at commit 11f9b58c0283586f4fb134ff8c022f1117b58223
- XMLAda from the public Git repository[3] at commit 8a9536bf161125cb1e12da376e8d7b51f1b33677

(I would have included GNATColl here, but it now relies on "libgpr"; waiting for GNAT GPL 2016 to see what this means.)

[1] http://www.gnu.org/licenses/gcc-exception-faq.html

[2] https://github.com/AdaCore/gprbuild

[3] https://github.com/AdaCore/xmlada

This is GCC 6.1.0, rebuilt as a cross-compiler from Mac OS X to arm-eabi (specifically, the Cortex-M3 as found on the Arduino Due[1] and the Cortex-M4 as found on the STMicroelectronics[2] STM32F4 Discovery and STM32F429I Discovery boards).

The compiler comes with no Ada Runtime System (RTS). See the Cortex GNAT Run Time Systems project[3] for candidates.

[1] http://www.arduino.com

[2] http://www.st.com

[3] https://sourceforge.net/projects/stm32f4-gnat-rts/

From: Simon Wright <simon@pushface.org>
Date: Sun, 05 Jun 2016 11:04:58 +0100
Subject: Re: ANN: GCC 6.1.0 for OS X El Capitan
Newsgroups: comp.lang.ada

It turns out that the native-2015 README makes a claim that I in fact failed to fulfil. I've updated it and uploaded a patch for manual application, if needed; only if you're going to be building a relocatable library (e.g. your own version of GNATColl).

The relevant section of the updated README says

Library names

The ld error "library not found for -lgnat-6.1" will be encountered when using gprbuild to link a relocatable library (ordinary relocatable links should be OK). The reason is that the GCC build process only generates libgnat-6.dylib, libgnarl-6.dylib, but gprbuild tries to link against libgnat-6.1.dylib (why it wouldn't just link against libgnat.dylib I don't know).

You can fix by either of

* installing the necessary symbolic links:
  in $prefix/lib/gcc/x86-64-apple-darwin15/6.1.0/adalib,
  sudo ln -s libgnat-6.dylib libgnat-6.1.dylib
  sudo ln -s libgnarl-6.dylib libgnarl-6.1.dylib

* patch gprbuild's database: download share-gprconfig-compilers.xml.diff, then in $prefix,
  sudo patch -p1 <~/Downloads/share-gprconfig-compilers.xml.diff

[See also " Mac OS X: GCC", AUJ 36-4, p. 206. —sparre]

From: Simon Wright <simon@pushface.org>
Date: Sun, 03 Jul 2016 18:07:05 +0100
Subject: ANN: GCC 6.1.0 for OS X El Capitan, with GPL 2016 tools
Newsgroups: comp.lang.ada

At https://sourceforge.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/6.1.0/native-2016/

This is GCC 6.1.0 built for Mac OS X El Capitan (10.11.4, Darwin 15.4.0), with the Command Line Tools for Xcode 7.

gcc-6.1.0-x86_64-apple-darwin15-2016-bin.tar.bz2

Compilers included: Ada, C, C++, Objective C, Objective C++, Fortran.

Tools included:

Full GPL:

- ASIS, AUnit, and GDB from GNAT GPL 2016.

- Gprbuild from the public Git repository[2] at commit c5c26c2683fcdd9c1fc684274faaad1b30c762e1

GPL with Runtime Library Exception[1]:

- GNATCOLL from the public GIT repository[3] at commit 719fae1d0b60c31a9f7c0de8b0a143fa57449b47

- XMLAda from the public Git repository[4] at commit a9536bf161125cb1e12da376e8d7b51f1b33677

[1] http://www.gnu.org/licenses/gcc-exception-faq.html

[2] https://github.com/AdaCore/gprbuild

[3] https://github.com/AdaCore/gnatcoll

[4] https://github.com/AdaCore/xmlada

Mac OS X: GCC for ARM-EABI

From: Simon Wright <simon@pushface.org>
Date: Mon, 23 May 2016 16:03:20 +0100
Subject: ANN: GCC 6.1.0 arm-eabi for OS X El Capitan
Newsgroups: comp.lang.ada

This is available at [1].

This is GCC 6.1.0, rebuilt as a cross-compiler from Mac OS X to arm-eabi (specifically, the Cortex-M3 as found on the Arduino Due[1] and the Cortex-M4 as found on the STMicroelectronics[2] STM32F4 Discovery and STM32F429I Discovery boards).

The compiler comes with no Ada Runtime System (RTS). See the Cortex GNAT Run Time Systems project[3] for candidates.

The compiler is known to run on El Capitan; it may not run on earlier OS X releases.

** FOR THE BENEFIT OF THE 15 PEOPLE WHO DOWNLOADED IT ALREADY **

If you got it to work, congratulations!

Ticket 17[2] says

Because of a stupid PATH setting error and not checking properly, the gcc-6.1.0-arm-eabi-bin.tar.bz2 package with MD5 2c8cdc7b032c7305faa05ed8f84f9f20 won't run after installation (it's looking for libstdc++.dylib somewhere it won't exist on your machine, fails to find it, uses the /usr/lib version, crashes out because of missing symbol).

A corrected version is available, MD5 83eef4a5358d5764379e2c7a74f7f2a0.

[1] https://sourceforge.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/6.1.0/arm-eabi/

[2] https://sourceforge.net/p/gnuada/bugs/17/

[See also " Mac OS X: GCC for ARM-EABI", AUJ 37-1, p. 18. —sparre]

Mac OS X: GNAT GPL for ARM-EABI

From: Simon Wright <simon@pushface.org>
Date: Mon, 06 Jun 2016 20:34:35 +0100
Subject: ANN: GNAT GPL 2016 arm-eabi for OS X El Capitan
Newsgroups: comp.lang.ada

This release at [1].

This is GNAT GPL 2016, rebuilt as a cross-compiler from Mac OS X to arm-eabi. The CPUs supported include cortex-m3, cortex-m4, cortex-r4.

The runtimes from the AdaCore gnat-gpl-2016-arm-elf-linux-bin are included:

- ravenscar-full-stm32f4

- ravenscar-full-stm32f429disco

- ravenscar-full-stm32f469disco

- ravenscar-full-stm32f7disco

- ravenscar-full-tms570

- ravenscar-sfp-stm32f4

- ravenscar-sfp-stm32f429disco

- ravenscar-sfp-stm32f469disco

- ravenscar-sfp-stm32f7disco

- ravenscar-sfp-tms570

- zfp-lm3s

- zfp-stm32f4

- zfp-tms570

as are the examples in share/examples/gnat-cross/.

[1] https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2016-arm-eabi-darwin-bin/

[See also " Mac OS X: GNAT for ARM-EABI", AUJ 36-3, p. 126. —sparre]

References to Publications

Books

From: Michael Vinn <ashos.owner@gmail.com>
Date: Thu, 10 Mar 2016 07:00:29 -0800
Subject: Looking for better Ada books
Newsgroups: comp.lang.ada

Are there any better books out there!

I have the following books.

+ Programming with ada by Peter Wegner

+ Programming in Ada by Barnes 3rd edition

+ Programming in Ada 95 by John Barnes 2nd edition

+ Software Components with Ada by Grady Booch

+ Reference Manual for the Ada Programming Language 1983

+ Ada 95 Reference Manual

+ Ada 95 Rationale

+ Understanding Ada by Bray and Pokrass

From: Anh Vo <anhvofrcaus@gmail.com>
Date: Thu, 10 Mar 2016 08:59:40 -0800
Subject: Re: Looking for better Ada books
Newsgroups: comp.lang.ada

[...]

You are way behind technologies for sure. I suggest that you move to the latest Ada, ISO/IEC 8652:2012(E) http://www.adaic.org/resources/add_content/standards/12rm/html/RM-TTL.html.

In addition, take a look at Ada 2012 Rationale http://www.ada-auth.org/standards/12rat/html/Rat12-TTL.html.

From: Olivier Henley <olivier.henley@gmail.com>
Date: Thu, 10 Mar 2016 09:35:17 -0800
Subject: Re: Looking for better Ada books
Newsgroups: comp.lang.ada

1. Ada for Software Engineers 2nd ed. 2009 Edition, ISBN-13: 978-1848823136

2. Concurrent and Real-Time Programming in Ada 3rd Edition, ISBN-13: 978-0521866972

3. Programming in Ada 2012 1st Edition, ISBN-13: 978-1107424814

... were excellent. (Have not finished 3. yet, but amazing so far)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 10 Mar 2016 15:39:12 -0600
Subject: Re: Looking for better Ada books
Newsgroups: comp.lang.ada

> [...]

Sure seems like my library of dusty old Ada books. Most of which I haven't opened in decades... :-)

[...]

But it's silly these days to restrict oneself to "books". There are a number of electronic resources to use as well (Ada Distilled - a PDF "book", and the Ada wikibooks site come to mind) - which have the distinct advantage of being free.

For a rather inclusive list of current Ada learning materials, see

http://www.adaic.org/learn/materials/.

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Fri, 11 Mar 2016 14:05:02 -0000
Subject: Re: Looking for better Ada books
Newsgroups: comp.lang.ada

> [...]

Building Parallel, Embedded and Real Time Applications with Ada McCormick, Singhoff and Hughes,

And at least one book covering SPARK.

John Barnes has one, "High Integrity Software" is good but covers an older version of SPARK (the book has been updated, but I haven't seen the newer version)

In my in-tray waiting is "Building High Integrity Applications with SPARK" (McCormick and Chapin).

[See also “Books for Learning Ada”, AUJ 35-2, p. 84. —sparre]

Book: Ada and SPARK on ARM Cortex-M

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 30 Mar 2016 06:13:15 -0700
Subject: Ada on ARM Cortex-M
Newsgroups: comp.lang.ada

I'm pleased to announce that the tutorial titled "Ada and SPARK on ARM Cortex-M" got its second edition:

http://inspirel.com/articles/Ada_On_Cortex.html

The tutorial was extended and now targets four popular development boards:

- Arduino M0 (or Genuino Zero)
- Arduino Due
- STM32 Nucleo-32 (with F0 chip)
- STM32 Nucleo-144 (with F7 chip)

The tutorial explains how to write Ada/SPARK programs with zero run-time, without any underlying layers and based solely on the information from chip reference documentation.

Source code for all examples, for all of these boards, is also available.

[See also “ Book: Ada and SPARK on ARM Cortex-M”, AUJ 36-3, p. 128. —sparre]

Ada Inside

Lisp Interpreter

From: Chris Moore <zmower@ntlworld.com>
Date: Fri, 25 Mar 2016 20:33:32 +0000
Subject: Make A Lisp .. in Ada
Newsgroups: comp.lang.ada

There was a post on Hacker News (#1) just over a year ago about Make A Lisp (#2). They had implementations in many languages but not for Ada so I've made one (#3 on branch ada).

I'm pretty close to getting it accepted (put in the pull request today). It's slower than the C and C++ implementations. Seems to spend a lot of time finalizing the smart pointer according to gprof.

1) https://news.ycombinator.com/item?id=9121448

2) https://github.com/kanaka/mal

3) https://github.com/kanaka/mal/tree/master/ada

MAX! Home Automation

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 14 Apr 2016 18:53:37 +0200
Subject: MAX! home automation 1.3 released
Newsgroups: comp.lang.ada

MAX! home automation is a GTK+ application to manage ELV/eQ-3 MAX! cubes. A cube is a gateway to a network of radiator thermostats, shutter contacts etc.

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

Changes to the previous version:

- MQTT server added;
- LAN discovery by default scans all known interfaces;
- Settings page allows explicit setting of the host address to scan or cube address to connect.

[See also “ MAX! Home Automation”, AUJ 36-4, p. 207. —sparre]

Mine Detector

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Fri, 6 May 2016 16:19:21 -0700
Subject: Improved Mine Detector Available
Newsgroups: comp.lang.ada

Pascal Malaise has contributed a change to the GTKAda versions of Mine Detector to preserve the appearance of the mine field when a game is not in progress. He also helped identify an improvement that makes the game significantly faster, which also applies to the Gnoga version. The new versions are available on the PragmAda web site; the Gnoga version is also available at

https://github.com/jrcarter

[See also “ Mine Detector”, AUJ 37-1, p. 18. —sparre]

Telesoft Telegen Expertise Needed

From: Rick Cottle <rdc2732@gmail.com>
Date: Wed, 15 Jun 2016 14:12:55 -0700
Subject: Help Needed - Telesoft Telegen 3.2.5 expert VAX/VMS
Newsgroups: comp.lang.ada

We need to touch an old project and of course there is very little documentation available (or readily found) and previous contributors move on a long time ago.

In short, the project is an embedded system built with Mil-Std-1750 built with ADA on a Vax. We have recreated the project to the point of trying to link but do not have a linker map to go the final step.

It would be awesome to find copies of the subject manuals. Additionally someone experienced in this environment for consulting work. Phoenix AZ area.

Ada in Context

Uninitialized “out” Parameters

From: Ahlan Marriott <ahlan@marriott.org>
Date: Tue, 5 Apr 2016 05:02:49 -0700
Subject: Uninitialized out parameters.
Newsgroups: comp.lang.ada

Is this a GNAT (GPL-2015) bug or my not understanding Ada?

I was surprised that I could compile:

```
procedure Test (V : out Positive) is null;
```

And even more by the results of calling the procedure:

```
   V : Positive;
begin
   Test (V);
   Ada.Text_IO.Put_Line ("V:" & V'img);
```

The value zero is output, which because V is positive should be impossible.

I would have thought that null procedures without parameters would fail to compile.

Opinions anyone?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 6 Apr 2016 15:47:49 -0500
Subject: Re: Uninitialized out parameters.
Newsgroups: comp.lang.ada

> [...]

I agree with Georg here. It *seems* like checks like the one Ahlan is suggesting are a good idea, until you trip over one. (The check in Ada that every function have at least one return is a similar idea, which causes no end of trouble.)

I can think of at least three reasons why one might write a null procedure with an out parameter:

(1) The null procedure body is a TBD placeholder. It will be replaced with a real body at some future point, but we still want to compile.

(2) The out parameter isn't used for some implementations. This often comes up when there are multiple parameters. (We ran into this commonly in Claw, although we usually used in out parameters in such cases to avoid de-initializing objects.) The situation is that some objects need additional return information and others don't:

```
procedure Do_Something (
      Obj : in out Object;
      Result : in out Result_Type;
      Extra_Info : out Natural);
```

Extra_Info is only used if Result has a particular value.

(3) The null procedure is used in an interface. In that case, giving a body isn't possible.

I think you could make a case that all of these are better written some other way, but that's irrelevant, in that each of these could happen in real code, and making a legality check would break that code. (Thus, making it illegal would probably be considered too incompatible for future Ada, unless of course we discovered some semantic problem that doing that would fix.)

The Ada Standard has nothing to say about warnings (other than for pragmas). Perhaps some future version of Ada will change that, but as of now, every warning is implementation-defined, and thus it they don't have anything to do with the language. (That is, talk to your vendor about warnings.)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 6 Apr 2016 15:54:44 -0500
Subject: Re: Uninitialized out parameters.
Newsgroups: comp.lang.ada

[...]

The rules for whether an out parameter is initialized (and if so, how) are complicated - see 6.4.1(12-15). 6.4.1(15) most likely applies in this case (assuming no Default_Value aspect is involved), and that says that the value is uninitialized. And of course the value of an uninitialized object can be anything.

Portable Lengths of Standard.String

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 13 Apr 2016 16:29:27 -0500
Subject: Re: Substrings as argument to procedures/functions
Newsgroups: comp.lang.ada

> [...]

Probably a better example is to remember that the range of Positive is implementation-defined, and that the language only requires the upper bound to be at least 32767. So if you need strings that have potentially more characters than that (to read an entire text file, for instance), and you want the code to be unconditionally portable (to steal someone else's line), you need to declare a type yourself (here, assuming that a million characters are enough):

```
type Big_Natural is range 0 .. 1_000_000;
subtype Big_Positive is Big_Natural
      range 1 .. Big_Natural'Last;
type Big_String is array (Big_Positive
      range <>) of Character;
```

You can do almost anything you can do with a String with a Big_String (and you can convert a Big_String to a String so you can use Put_Line and the like), but the index type is guaranteed to support up to a million characters.

From: Robert I. Eachus <rieachus@comcast.net>
Date: Mon, 25 Apr 2016 08:33:18 -0700
Subject: Re: Substrings as argument to procedures/functions
Newsgroups: comp.lang.ada

Is it time to "fix" this? Or are there still Ada compilers around that use 16-bit String indexes by default? I remember when (in the early days of Ada 83) the question of whether String should use 16 or 32 bit indexes was a major implementation decision. Today, compilers that support hardware indexes shorter than 32-bits are probably mapping to some hardware defined offset field in instructions. Using such indexes when the subtype allows is obviously an optimization worth supporting. But in Ada defining "subtype Short_String is String range (1..32767);" or whatever allows using such a hardware type if available. But there is no need to make Short_String a type.

Yes, programs may have records with character string fields limited such that the index can fit into a 16-bit field in the record. But I can't imagine doing that without an explicit layout for the record or at least a size for the index field. Back to the original topic here, you want/need sliding to work such that

```
type Rec is
   Name_Length : Integer range 0..32_767;
   Name: String;
end;
for Rec use ...
...
Name_Length := Param'Length;
Name (1 .. Name_Length) := Param;
```

works as expected.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 25 Apr 2016 17:07:02 -0500
Subject: Re: Substrings as argument to procedures/functions
Newsgroups: comp.lang.ada

> Is it time to "fix" this?

Use of predefined types is evil, so who cares?

> Or are there still Ada compilers around that use 16-bit String indexes by default?

Janus/Ada for one.

All versions of Janus/Ada have had Integer as 16-bit. Changing that would destroy compatibility of binary files (Sequential_IO, Direct_IO, Stream_IO) and of course would have other effects as well.

I've considered having some sort of optional way to change the definition of Integer (it's defined in a single place), but the problem is that *everything* depends upon that, so one would end up with two different incompatible compilers/runtimes. The maintenance headaches would be immense (any mix-up would cause bizarre internal errors).

Good code doesn't depend on predefined types in the first place, so it's mainly a problem dealing with the predefined packages. (And that's mainly a problem with the predefined packages depending on type String.)

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Tue, 26 Apr 2016 08:12:46 +0200
Subject: Re: Substrings as argument to procedures/functions
Newsgroups: comp.lang.ada

> Use of predefined types is evil, so who cares?

Anyone coming from just about any other language, since they have no idea that there could be such a fine thing as user defined fundamental types! ;-)

So, how does one introduce them to programming in Ada without using library types but still using strings, and text I/O?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 26 Apr 2016 13:41:53 -0500
Subject: Re: Substrings as argument to procedures/functions
Newsgroups: comp.lang.ada

> So, how does one introduce them to programming in Ada without using library types but still using strings, and text I/O?

Using Root_String'Class, of course. Which Ada doesn't have, unfortunately. I'd rather fix that rather than noddling with Standard types - the Wide_Wide_Unbounded_UTF8_String nonsense doesn't make sense (but maybe it's UTF8_Unbounded_Wide_Wide_String? Bah hambug.)

Task_Termination and Tasks in the Compiler Run-time

From: Per Dalgas Jakobsen <pdj@knaldgas.dk>
Date: Thu, 21 Apr 2016 12:23:50 +0200
Subject: timer_server triggers Task_Termination handler
Newsgroups: comp.lang.ada

Is it correct behaviour when tasks internal to the GNAT run-time causes users task_termination handlers to be called?

This behaviour is seen on:

1) Debian Linux: gnat-5 (Ada 2005, Ada 2012).

2) AIX: GNAT Pro 6.1.0w (Ada 2005).

A simple demonstration of the issue:

```
with Ada.Text_IO;
with Log_Unhandled_Exceptions;

procedure Timer_Server_Noise is

begin
   Ada.Text_IO.Put_Line ("Start of main");

   select
      delay 0.5;
   then abort
      loop
         delay 0.1;
      end loop;
   end select;

   Ada.Text_IO.Put_Line ("End of main");
end  Timer_Server_Noise;

with Ada.Exceptions;
with Ada.Task_Identification;
with Ada.Task_Termination;

package Log_Unhandled_Exceptions is
   pragma Elaborate_Body;

   use Ada.Task_Identification;
   use Ada.Task_Termination;
   use Ada.Exceptions;

   protected Last_Wishes is
      procedure Log_Any_Exit (Cause : in
            Cause_Of_Termination;
            T : in Task_Id;
            E : in Exception_Occurrence);
   end;

end Log_Unhandled_Exceptions;

with Ada.Text_IO;
package body Log_Unhandled_Exceptions
is

   -- Encapsulates the actual log call
   procedure Log (Text : in String) is
   begin
      Ada.Text_IO.Put_Line
            ("Log_Unhandled_Exceptions
            >> " &  Text);
   end Log;

   protected body Last_Wishes is

      procedure Log_Any_Exit (Cause : in
            Cause_Of_Termination;
            T : in Task_Id;
            E  : in Exception_Occurrence) is
      begin
         case Cause is
            when Normal =>
               Log ("Normal exit of task: " &
               Image (T));
            when Abnormal =>
               Log ("Abnormal exit of task: " &
               Image (T));
            when Unhandled_Exception =>
               Log ("Unhandled exception in task:
               " & Image (T));
         end case;
      end Log_Any_Exit;

   end Last_Wishes;

begin
   if Current_Task_Fallback_Handler = null
then
      Set_Dependents_Fallback_Handler
         (Last_Wishes.Log_Any_Exit'Access);
   else
      Log ("Fallback handler already set, will
            not set own handler.");
   end if;

   if Specific_Handler (Current_Task) = null
then
      Set_Specific_Handler (Current_Task,
         Last_Wishes.Log_Any_Exit'Access);
   else
      Log ("Specific handler already set, will
not set own handler.");
   end if;
end Log_Unhandled_Exceptions;
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 21 Apr 2016 16:13:45 -0500
Subject: Re: timer_server triggers Task_Termination handler
Newsgroups: comp.lang.ada

> Is it correct behaviour when tasks internal to the GNAT run-time causes users task_termination handlers to be called?

Sure, why not?

In your example, you set a handler for all dependent tasks of the environment task (that is, ALL tasks). C.7.3 doesn't specify who wrote the task or for what purpose. If it is a dependent of the environment task, your handler will be called.

We've always encouraged implementers to write some or all of their runtime in Ada. It would increase the difficulty quite a bit if one had to "cover up" the effects of using Ada to write the code.

And I can't quite imagine what rule one write to exclude tasks that happen to be in library code. If one said to exclude tasks only if they are in language-defined packages, then you'll still get the tasks that happen to occur in implementation-defined stuff. And since that sort of stuff underlies many language-defined packages, and often is visible to the user as well, how do you account for tasks in such implementation-defined packages. (Surely I hope it doesn't depend on how the package is used!)

And what about tasks in third-party libraries? Claw, for instance, includes a hidden task. Should that be excluded? Should it be excluded only in Janus/Ada (where Claw is not a third-party library) but included in GNAT (where Claw is essentially user code)? That way seems to lead to madness.

Rereading some of the mail on the original AI, part of the intent was that one could set a handler on ALL tasks, including those not visible to the programmer (hidden in package bodies). If you can see the tasks and only want specific tasks involved, specific handlers make more sense. So in summary I believe this is working the way it was intended. Perhaps we should have put a note in that "all descendent tasks" include tasks that aren't visible to the programmer (hidden in the runtime or third-party packages), but clearly the idea was that the handler would work on every task in the partition, no matter what is its source.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Fri, 22 Apr 2016 07:41:23 +0200
Subject: Re: timer_server triggers Task_Termination handler
Newsgroups: comp.lang.ada

> We've always encouraged implementers to write some or all of their runtime in Ada. It would increase the difficulty quite a bit if one had to "cover up" the effects of using Ada to write the code.

It could be considered user-friendly for hidden tasks in reusable components to specify a task specific handler, so that they remain hidden if the user specifies a general handler.

But I agree that it would be difficult to /require/ this in the ARM.-

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Fri, 22 Apr 2016 08:46:16 +0200
Subject: Re: timer_server triggers Task_Termination handler
Newsgroups: comp.lang.ada

>> Is it correct behaviour when tasks internal to the GNAT run-time causes users task_termination handlers to be called?

> Sure, why not?

Maybe because you end up having to inspect the run-time library to figure out why your application behaves like it does?

Or because it makes the behaviour of your program depend on which (correct) run-time library you compile it with?

> And what about tasks in third-party libraries?

In my opinion third-party libraries are a different matter from the run-time provided by the compiler.

I would definitely expect the handlers to be called for any tasks declared outside the run-time.

From: Robert A Duff <bobduff@TheWorld.com>
Date: Thu, 21 Apr 2016 17:26:46 -0400
Subject: Re: timer_server triggers Task_Termination handler
Newsgroups: comp.lang.ada

> Is it correct behaviour when tasks internal to the GNAT run-time causes users task_termination handlers to be called?

No. Internal tasks are an implementation detail, and should be invisible to Ada programs.

I fixed this bug in GNAT recently.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 22 Apr 2016 17:35:06 -0500
Subject: Re: timer_server triggers Task_Termination handler
Newsgroups: comp.lang.ada

> [...] Internal tasks are an implementation detail, and should be invisible to Ada programs.

Nice thought, but exactly the opposite to some of opinions in the e-mail associated with the design of the task termination feature. They wanted to be notified if an internal task failed (presumably to narrow down the cause of the inevitable failure cascade that follows).

Given the Ada definition, it is wrong to hide an Ada task. Of course, there is no reason to use Ada tasks in the runtime (the runtime could be written C or Prolog ;-), so there is no truely wrong answer here.

Broadcasting UDP

From: Ahlan Marriott <ahlan@marriott.org>
Date: Sun, 24 Apr 2016 09:31:48 -0700
Subject: Broadcasting UDP
Newsgroups: comp.lang.ada

I asked this question sometime ago but I can no longer find the post.

In any case it was never really resolved so let me try again.

I am looking for a platform independent Ada solution on how to broadcast a UDP packet.

I would be satisfied with a GNAT only solution, i.e. one that uses Gnat.Sockets and/or Gnat specific libraries.

Attempting to broadcast to the limited broadcast address 255.255.255.255 has no effect (under Windows at least)

To successfully broadcast one needs to use the subnet directed broadcast address.

As I want to support PCs that have multiple ethernet adapters this means that I must iterate over the ethernet adapters, discover my ethernet address and subnet for each adapter, calculate the broadcast address and then send the UDP packet to that address.

So far so good.

My problem is that I don't know how to iterate over my adapters and obtain the address and subnet mask for each adapter using just Ada.

Can anyone tell me how I can do this using Ada?

The solution I currently employ is to bind to a windows API.

However this is obviously not target independent.

If there is no Ada way of iterating over the ethernet adapters then I will have to do a separate implementation for each platform.

In which case I would be grateful if anyone could tell me how I can iterate my adapters under Linux and OSX.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 24 Apr 2016 19:22:39 +0200
Subject: Re: Broadcasting UDP
Newsgroups: comp.lang.ada

> [...]

```
declare
   Host : Host_Entry_Type :=
             Get_Host_By_Name (Host_Name);
   Address : aliased Sock_Addr_Type;
begin
   for Index in 1..Addresses_Length (Host)
loop
      Address.Addr := Addresses (Host,
            Index)));
      Address.Port := <port>;
      declare
         Socket : Socket_Type := No_Socket;
         Pier   : Sock_Addr_Type;
      begin
         Create_Socket (Socket, Family_Inet,
               Socket_Datagram);
         Set_Socket_Option (Socket,
               Socket_Level,
               (Reuse_Address, True));
         Set_Socket_Option (Socket,
               Socket_Level, (Broadcast, True));
         Set_Socket_Option (Socket,
               Socket_Level, (Receive_Timeout,
               <timeout>));
         Bind_Socket (Socket, Address);
         Address.Addr := Broadcast_Inet_Addr;
         Send_Socket (Socket, <request-
               packet>, Last, Address'Access);
         loop -- Collecting responses, time-
               limited <timeout>
            ...
            Receive_Socket (Socket, <response-
               packet>, Last, Pier);
            ...
         end loop;
         Close_Socket (Socket);
      end;
   end loop;
end;
```

> In which case I would be grateful if anyone could tell me how I can iterate my adapters under Linux

Under Linux it is through the proc file system, if I remember correctly.

An Element of a Coding Standard

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Thu, 28 Apr 2016 07:13:49 +0200
Subject: Re: Building an encapsulated library that uses GNAT sockets under Windows
Newsgroups: comp.lang.ada

[...]

when a coding standard says

"X is forbidden",

it really means

"X shall not be used, unless proper justification is given and approved by QA".

Or so should the coding standard say...

[See also " An Element of a Coding Standard", AUJ 36-3, p. 136. —sparre]

Preelaboration

From: Simon Wright <simon@pushface.org>
Date: Mon, 16 May 2016 17:26:58 +0100
Subject: Preelaboration
Newsgroups: comp.lang.ada

I (think I) need to eliminate elaboration calls in parts of my Cortex GNAT Runtime Systems project[1], and I'm left with one that I can't understand.

(1) what does pragma Preelaborate actually mean? I hoped it would mean "you don't need to elaborate this package".

(2) If you give Preelaborate, is it a compiler error to generate elaboration code? (the generated elaboration procedure does nothing).

(3) I find ARM 10.2.1(10)[2] unclear (I got here because the problematic package body instantiates a generic). Does it mean that the instantiation won't be preelaborable unless all of 10.1 .. 10.4 are false? Or does it mean that the compiler will make these assumptions regardless of actuals?

And, looking at (10.1), would you expect

```
generic
   type Item is private with
         Preelaborable_Initialization;

package Generic_Queues is
```

to be legal? GNAT rejects it, 'aspect "Preelaborable_Initialization" not allowed for formal type declaration'.

[1] https://sourceforge.net/projects/cortex-gnat-rts/

[2] http://www.ada-auth.org/standards/rm12_w_tc1/html/RM-10-2-1.html#p10

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Mon, 16 May 2016 11:22:29 -0700
Subject: Re: Preelaboration
Newsgroups: comp.lang.ada

> [...]

Preelaborable_Initialization is not listed as a language-defined aspect in ARM K.1. ARM 10.2.1 (11.8/2) says that pragma Preelaborable_Initialization may appear in a generic formal part, so if you replace the aspect with the pragma it should be legal.

From: Simon Wright <simon@pushface.org>
Date: Mon, 16 May 2016 20:55:45 +0100
Subject: Re: Preelaboration
Newsgroups: comp.lang.ada

> [...]

Thanks! Solved the compilation issue, but that wasn't the cause of the original problem.

I guess the fact that Preelaborable_Initialization isn't listed in K.1 was an oversight.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 16 May 2016 15:28:55 -0500
Subject: Re: Preelaboration
Newsgroups: comp.lang.ada

> (1) what does pragma Preelaborate actually mean? [...]

What it says in 10.2.1, no more and no less. There is no required effect on code generation (unless Annex C is supported, and even then it is limited).

> (2) If you give Preelaborate, is it a compiler error to generate elaboration code? [...]

No, the only errors are those defined by 10.2.1.

C.4 says that a subset of preelaborable packages should not generate any code. Which only applies if Annex C is implemented, and in any case is not a testable requirement (the ACATS cannot require code inspection, which is the only way to determine if C.4 is followed).

C.4(12) supposedly requires documentation of cases where code is generated, but that sort of requirement is widely ignored. (And would be useless, I'd just write something like "Elaboration of a preelaborable package may execute code in all cases other than those required by C.4 to not execute any code." It's way too hard for an implementer to figure out what will and will not do something.

(And this whole set of requirements is silly anyway. What implementer executes code if they don't have to?? So anything that can be done without code is done that way, and everything else executes some code, regardless of any categorization pragma.)

> (3) [...]

This means that the generic body would be illegal if preelaborated, unless it means the preelaboration requirements using those assumptions. The instance can't be preelaborable unless the body is.

> [...]

Preelaborable_Initialization is not an aspect, as it is not the same for all views of a type. We tried to come up with a model to make it an aspect and failed to come up with something that makes sense.

In addition, formal types aren't allowed to have language-defined aspects, as that would add implicit contracts to the specification (and notably, would need rules for those contracts). pragma P_I isn't allowed in a formal part, either.

To get a formal type that has P_I, you'd need to use a generic derived type where the ancestor has P_I. Probably not what you want.

I would guess in this case that the instance in a preelaborable package should make the package illegal; the generic would need to be Preelaborable, but then it most likely would be illegal.

But this area is complicated (I might be getting it all wrong) and it probably doesn't do what you want anyway (Pure/Preelaborable categorizations in general were a failure, as they can't be applied to the majority of packages). I don't really have any advice, other than if you really care, look up the old AIs on the topic.

From: Simon Wright <simon@pushface.org>
Date: Mon, 16 May 2016 22:03:47 +0100
Subject: Re: Preelaboration
Newsgroups: comp.lang.ada

[...]

Thanks for the advice. Things are clearer now, I think; anyway, it turns out it was indeed the generic that caused the package it was instantiated in to require elaboration. I expanded the generic (of which this was the only instance, huh) by hand in the caller, no more elaboration.

GNAT has a program-unit restriction No_Elaboration_Code[1] which is a lot closer to what I want. I can't remember now how I got into the state where the compiler told me I couldn't use it ...

[1] https://gcc.gnu.org/onlinedocs/gnat_rm/No_005fElaboration_005fCode.html

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 17 May 2016 18:25:47 -0500
Subject: Re: Preelaboration
Newsgroups: comp.lang.ada

> But 10.2.1(10.1), (11.2) explicitly mention P_I in a formal part?

These are bug, IMHO, because there are no matching rules for P_I. One could not be allowed to instantiate a generic whose formal has P_I with a type that does not have P_I. Else the assumptions of the body are violated.

> Besides, 10.2.1(11.6/2) says that the pragma has to appear in the visible part of a package or generic package; the formal part of a generic package is not part of the visible part.

I suspect that there was an intent that this would work, but there is a lot missing that would make it work.

From: Robert A Duff <bobduff@TheWorld.com>
Date: Tue, 17 May 2016 20:05:42 -0400
Subject: Re: Preelaboration
Newsgroups: comp.lang.ada

> Besides, 10.2.1(11.6/2) says that the pragma has to appear in the visible part of a package or generic package; the formal part of a generic package is not part of the visible part.

But 8.2(8) says otherwise. How else could it be? The generic formals have to be visible at the instantiation, so they can be referred to in named-notation assocs.

I'm not sure how this affects the P_I pragma, but for sure generic formals are visible at the instantiation.

Forced Compile-time Warning (GNAT)

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sun, 3 Jul 2016 09:05:39 +0300
Subject: Re: GNAT equivalent to gcc's #warning directive?
Newsgroups: comp.lang.ada

> I vaguely recall there was a gnat pragma to cause the compiler to spit out a user defined message. In C++ I often insert a cpp #warning when I need to go back and revisit some code. That way it nags me every time I build the project, until it gets addressed.

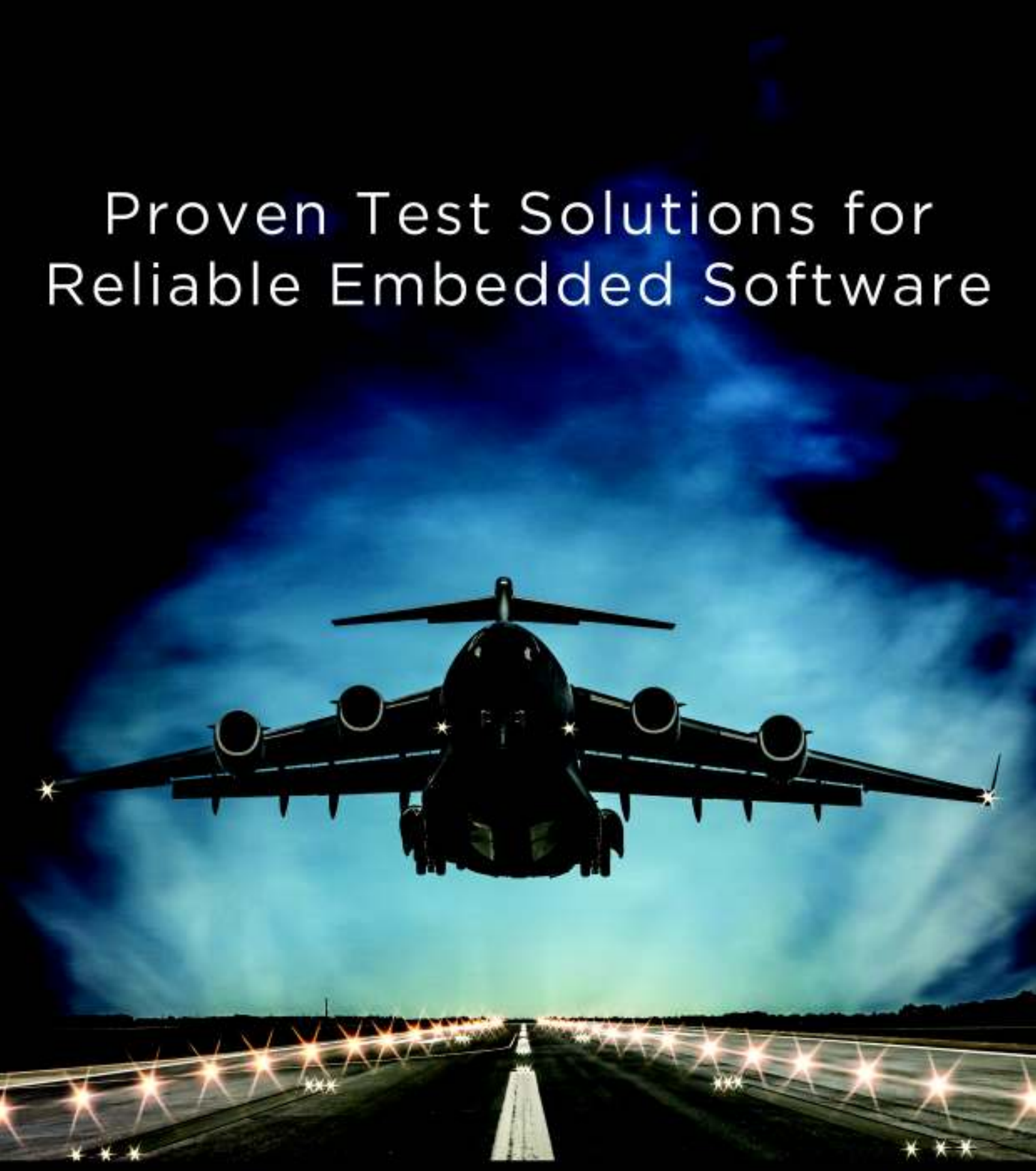
>

> I'm looking for something similar in gnat. I thought it was pragma Warnings(), but that appears to be for known compiler warnings.

Compile_Time_Warning:

http://docs.adacore.com/gnat_rm-docs/html/gnat_rm/gnat_rm/implementation_defined_pragmas.html#pragma-compile-time-warning

Proven Test Solutions for Reliable Embedded Software



"VectorCAST is unique in that it provides us with the ability to increase the reliability and quality of our flight software."

-- Honeywell

VECTOR software

vectorcast.com

VectorCAST is a TUV SUD Certified Software Tool for Safety Related Development

Vector Software, Inc.
Golden Cross House | 8 Duncannon Street | London WC2N 4JF UK | +44 203 603 0120 | sales@vectorcast.com

Conference Calendar

Dirk Craeynest
KU Leuven. 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.

2016

July 04-08 **Software Technologies: Applications and Foundations** (STAF'2016), Vienna, Austria. Successor of the TOOLS federated event. Topics include: practical and foundational advances in software technology, including formal foundations of software technology, testing and formal analysis, graph transformations and model transformations, model driven engineering, and tools.

- July 04-08 14th **International Conference on Software Engineering and Formal Methods** (SEFM'2016). Topics include: real-time, hybrid and embedded systems; verification and validation; light-weight and scalable formal methods; software evolution, maintenance and reuse; application and technology transfer; case studies, best practices and experience reports; tool integration; education; safety-critical, fault-tolerant and secure systems; software certification; programming languages; type theory; abstraction and refinement; etc.
- July 05-07 10th **International Conference on Tests And Proofs** (TAP'2016). Topics include: many aspects of verification technology, including foundational work, tool development, and empirical research; the connection between proofs (and other static techniques) and testing (and other dynamic techniques); verification and analysis techniques combining proofs and tests; program proving with the aid of testing techniques; deductive techniques to support testing: generating testing inputs and oracles, supporting coverage criteria, and so on; program analysis techniques combining static and dynamic analysis; testing and runtime analysis of formal specifications; model-based testing and verification; using model checking to generate test cases; testing of verification tools and environments; applications of testing and proving to new domains, such as security, configuration management, and language-based techniques; case studies, tool and framework descriptions, and experience reports about combining tests and proofs; etc.

☺ July 06-08 28th **Euromicro Conference on Real-Time Systems** (ECRTS'2016), Toulouse, France.

- July 05 12th **Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications** (OSPERT'2016). Topics include: providing a reliable and efficient operating environment for real-time and embedded applications, case studies and experience reports, certification and verification of RTOSs and middleware, real-time virtualization and hypervisors, RTOSs for manycore platforms, support for multiprocessor architectures, support for component-based development, etc.

July 11-13 9th **Seminar on Advanced Techniques & Tools for Software Evolution** (SATToSE'2016), Bergen, Norway. Topics include: all aspects of software and model evolution, practices and technologies; supporting tools, processes, and models for managing software evolution; industrial needs, case studies and experiences; empirical studies in evolution and maintenance; program transformation, refactoring, renovation and migration; reliability and security aspects of software (co-)evolution; software ecosystem evolution; etc.

July 17-19 10th **International Symposium on Theoretical Aspects of Software Engineering** (TASE'2016), Shanghai, China. Topics include: theoretical aspects of software engineering, such as abstract

interpretation, component-based systems, 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, object-oriented systems, run-time verification and monitoring, software architecture, software testing and quality assurance, software security and reliability, static analysis of programs, type systems and behavioural typing, tools exploiting theoretical results, etc.

☺ July 17-22 30th **European Conference on Object-Oriented Programming** (ECOOP'2016), Rome, Italy. Topics include: theory, design, implementation, optimization, and analysis of programming languages that enable or enforce abstractions across various programming styles, from object-orientation to reactivity to spreadsheets; innovative and creative solutions to real problems; evaluations of existing solutions in ways that shed new insights; etc.

- ☺ July 17 1st **Workshop on Programming Models and Languages for Distributed Computing** (PMLDC'2016). Topics include: new approaches to distributed programming that provide efficient execution and the elimination of accidental nondeterminism resulting from concurrency and partial failure.
- ☺ July 18 11th **Workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems** (ICOOOLPS'2016). Topics include: techniques for the implementation and optimization of a wide range of languages including but not limited to object-oriented ones; implementation and optimization of fundamental languages features (from automatic memory management to zero-overhead metaprogramming); runtime systems technology; compilers (intermediate representations, offline and online optimizations, ...); empirical studies on language usage; resource-sensitive systems (real-time, low power, mobile, cloud); tooling support, debuggability and observability of languages as well as their implementations; etc.
- July 18 **Workshop on Programming Experience** (PX'2016). Topics include: exploratory programming, navigation, modularity mechanisms, literacy, tool building, language engineering, etc.
- July 18 1st **Workshop on Runtime Verification for Object-Oriented Languages, and Systems** (VORTEX'2016). Topics include: combination of static and dynamic analyses, industrial applications, monitoring concurrent/distributed systems, RV for safety and security, tool development, etc.
- July 19 18th **Workshop on Formal Techniques for Java-like Programs** (FTfJP'2016). Topics include: language semantics, specification techniques and languages, verification of program properties, verification logics, dynamic program analysis, static program analysis, type systems, security.

July 17-23 28th **International Conference on Computer Aided Verification** (CAV'2016), Toronto, Ontario, Canada. Topics include: theory and practice of computer-aided formal analysis methods for hardware and software systems, algorithms and tools for verifying models and implementations, program analysis and software verification, verification methods for parallel and concurrent systems, testing and run-time analysis based on verification technology, applications and case studies in verification, verification in industrial practice, formal models and methods for security, etc.

July 24-26 11th **International Joint Conference on Software Technologies** (ICSOFT'2016), Lisbon, Portugal. Topics include: all areas that are either related to new software paradigm trends or to mainstream software engineering and applications, such as software metrics, agile methodologies, risk management, quality control and assurance, software standards and certification, software and systems integration, software testing and maintenance, model-driven engineering, software and systems quality, software and information security, formal methods, programming languages, middleware technologies, parallel and high performance computing, etc.

July 25-28 35th **Annual ACM Symposium on Principles of Distributed Computing** (PODC'2016), Chicago, Illinois, USA.

August 01-03 IEEE **International Conference on Software Quality, Reliability and Security** (QRS'2016), Vienna, Austria. Merger of SERE (International Conference on Software Security and Reliability) and QSIC (International Conference on Quality Software) Topics include: reliability, security, availability, and safety of software systems; software testing, verification and validation; metrics, measurements, and analysis; software vulnerabilities; formal methods; benchmark, tools, and empirical studies; etc.

Includes IEEE International Workshop on Safety and Security in Cyber-Physical Systems (SSCPS), on Trustworthy Computing (TC), etc.

August 02-05 11th IEEE **International Conference on Global Software Engineering** (ICGSE'2016), Orange County, California, USA. Theme: "Software Bridging Distances Between People". Topics include: industrial offshoring and outsourcing experiences, lean and agile development, methods and processes, mining software repositories and software analytics, open source software communities, security and privacy, software evolution and maintenance, strategic issues in distributed development, tools and infrastructure support, etc.

Aug 31 - Sep 02 42nd **Euromicro Conference on Software Engineering and Advanced Applications** (SEAA'2016), Limassol, Cyprus. Topics include: information technology for software-intensive systems; embedded software engineering (ESE); model-based development, components and services (MOCS); software process and product improvement (SPPI); teaching, education and training for dependable embedded and cyberphysical systems (TET-DEC); cyber-physical systems (CPS).

September 03-07 31st IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2016), Singapore. Topics include: foundations, techniques, and tools for automating the analysis, design, implementation, testing, and maintenance of large software systems, such as component-based systems, maintenance and evolution, model-driven development, model transformations, modeling language semantics, open systems development, re-engineering, specification languages, software architecture and design, software product line engineering, testing, verification, and validation, etc.

September 05-09 12th **European Dependable Computing Conference** (EDCC'2016), Gothenburg, Sweden. Topics include: theory, techniques, systems, and tools for the design, validation, operation and evaluation of dependable and secure computing systems, covering any fault model, from traditional hardware and software faults to accidental and malicious human interactions; dependability in practice (industrial applications, experience in introducing dependability in industry, use of new or mature dependability approaches to new challenging problems or domains, ...); hardware and software architecture of dependable systems; safety critical systems; embedded and real-time systems; cyber-physical systems; testing and validation methods; security of systems and networks; etc.

September 05-06 8th **International Workshop on Software Engineering for Resilient Systems** (SERENE'2016). Topics include: requirements engineering & re-engineering for resilience; frameworks, patterns and software architectures for resilience; design of trustworthy systems; verification, validation and evaluation of resilience; empirical studies in the domain of resilient systems; methodologies adopted in industrial contexts; resilient cyber-physical systems; etc.

September 08-09 10th ACM/IEEE **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2016), Ciudad Real, Spain. Topics include: strengths and weaknesses of software engineering technologies and methods from a strong empirical viewpoint, including quantitative, qualitative, and mixed studies; empirical studies using qualitative, quantitative, and mixed methods; case studies, action-research, and field studies; replication of empirical studies and families of studies; empirically-based decision making; mining software engineering repositories; assessing the benefits / costs associated with using certain development technologies; industrial experience, software project experience, and knowledge management; software technology transfer to the industry; etc.

September 11-14 **Federated Conference on Computer Science and Information Systems** (FedCSIS'2016), Gdansk, Poland. Topics include: education, curricula & research methods; software systems development & applications; etc.

September 12-14 15th **International Conference on Intelligent Software Methodologies, Tools and Techniques** (SoMeT'2016), Larnaca, Cyprus. Topics include: state-of-art and new trends on software methodologies, tools and techniques; software methodologies and tools for robust, reliable, non-fragile software design; software development techniques for legacy systems; software evolution techniques; agile software and lean methods; formal methods for software design; software maintenance; software security tools and techniques; formal techniques for software representation, software testing and validation; object-oriented, aspect-oriented, component-based and generic programming, multi-agent technology; Model Driven Development (DVD), code centric to model centric software engineering; etc.

September 23-30 16th **International Conference on Runtime Verification** (RV'2016), Madrid, Spain. Topics include: monitoring and analysis of software and hardware system executions. Application areas include: cyber-

physical systems, safety/mission-critical systems, enterprise and systems software, autonomous and reactive control systems, health management and diagnosis systems, and system security and privacy.

☺ September 26-29 21st **International Workshop on Formal Methods for Industrial Critical Systems** & 16th **International Workshop on Automated Verification of Critical Systems** (FMICS-AVoCS'2016), Pisa, Italy. Topics include: design, specification, refinement, code generation and testing of critical systems based on formal methods; methods, techniques and tools to support automated analysis, certification, debugging, learning, optimization and transformation of critical systems, in particular distributed, real-time systems and embedded systems; automated verification (model checking, theorem proving, SAT/SMT constraint solving, abstract interpretation, etc.) of critical systems; verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability (e.g., scalability and usability issues); tools for the development of formal design descriptions; case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; impact of the adoption of formal methods on the development process and associated costs; application of formal methods in standardization and industrial forums. Deadline for submissions: August 17, 2016 (research ideas).

September 26-29 35th **International Symposium on Reliable Distributed Systems** (SRDS'2016), Budapest, Hungary. Topics include: dependability in cyber-physical systems, distributed objects and middleware systems, experimental or analytical evaluations of dependable distributed systems, formal methods and foundations for dependable distributed computing, high-assurance and safety-critical distributed system design and evaluation, secure and trusted distributed systems, etc.

October 02-10 32nd **International Conference on Software Maintenance and Evolution** (ICSME'2016), Raleigh, North Carolina, USA. Topics include: reverse engineering and re-engineering, software refactoring and restructuring, software migration and renovation, software and system comprehension, software repository analysis and mining, software testing, maintenance and evolution processes, software quality assessment, continuous integration/deployment, etc. Deadline for early registration: August 10, 2016.

October 06 6th **International Workshop on Design, Modeling and Evaluation of Cyber Physical Systems** (CyPhy'2016), Pittsburgh, Pennsylvania, USA. In conjunction with ESWEEK 2016. Topics include: development of industrial or research oriented cyber-physical systems in domains such as robotics, smart systems (homes, vehicles, buildings), medical and healthcare devices, future generation networks; evaluation of novel research tools; comparisons of state of the art tools in industrial practice; etc. Deadline for submissions: July 10, 2016.

☺ October 06-07 ACM SIGAda's **High Integrity Language Technology International Workshop on Model-Based Development and Contract-Based Programming** (HILT'2016), Pittsburgh, Pennsylvania, USA. Sponsored by ACM SIGAda. Co-located with EMSOFT 2016 (ACM SIGBED's International Conference on Embedded Software), part of ESWeek 2016 (Embedded Systems Week). Topics include: automated analysis and code generation targeting verification-oriented tools and/or programming language subsets (such as SPARK/Ada, ...); contributions linking modeling and contracts to the topics associated with the co-located EMSOFT conference (such as model- and component-based software design and analysis, software technologies for safety-critical and mixed-critical systems, robust implementation of control systems, ...); etc. Deadline for submissions: July 15, 2016 (papers, extended abstracts).

☺ Oct 30 - Nov 11 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2016), Amsterdam, the Netherlands. Topics include: all aspects of software construction, at the intersection of programming, languages, systems, and software engineering. Deadline for submissions: July 8, 2016 (posters), August 15, 2016 (Student Research Competition).

☺ November 01 **High Integrity Software Conference** (HIS'2016), Bristol, UK.

December 10 Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

2017

☺ January 18-20 44th ACM SIGPLAN **Symposium on Principles of Programming Languages** (POPL'2017), Paris, France. Topics include: all aspects of programming languages and programming systems. Deadline for submissions: July 6, 2016 (papers).

February19-21 5th **International Conference on Model-Driven Engineering and Software Development** (MODELSWARD'2017), Porto, Portugal. Topics include: domain-specific modeling, general-purpose modeling languages and standards, syntax and semantics of modeling languages, model-based testing and validation, model execution and simulation, model quality, component-based software engineering, software factories and software product lines, etc. Deadline for submissions: October 7, 2016 (regular papers), November 10, 2016 (workshops), November 11, 2016 (position papers), November 28, 2016 (special session), December 14, 2016 (doctoral consortium), January 3, 2017 (tutorials, demos, panels).

April 3-7 32nd ACM **Symposium on Applied Computing** (SAC'2017), Marrakech, Morocco.

☺ April 3-7 **Track on Object-Oriented Programming Languages and Systems** (OOPS'2017). Topics include: aspects and components; code generation, and optimization; distribution and concurrency; formal verification; integration with other paradigms; interoperability, versioning and software evolution and adaptation; language design and implementation; modular and generic programming; runtime verification; secure and dependable software; static analysis; testing and debugging; type systems; Virtual machines; etc. Deadline for submissions: September 15, 2016 (regular papers, Student Research Competition abstracts).

☺ May 20-28 39th **International Conference on Software Engineering** (ICSE'2017), Buenos Aires, Argentina. Deadline for submissions: August 26, 2016 (technical research papers); October 7, 2016 (workshop proposals); October 26, 2016 (Software Engineering in Practice, Software Engineering Education & Training, New Ideas and Emerging Results, Software Engineering in Society); November 18, 2016 (formal demonstrations, technical briefings, Doctoral Symposium); December 28, 2016 (Student Research Competition); January 9, 2017 (posters).

♦ June 12-16 22nd **International Conference on Reliable Software Technologies - Ada-Europe'2017**. Vienna, Austria. Sponsored by Ada-Europe. Deadline for submissions: January 15, 2017 (papers, tutorials, workshops, industrial presentations).

December 10 Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

Conference Chair

Wolfgang Kastner
TU Vienna, Austria

Program Co-Chairs

Johann Blieberger
blieb@auto.tuwien.ac.at
TU Vienna, Austria

Tullio Vardanega
tullio.vardanega@math.unipd.it
Università di Padova, Italy

Special Session Chair

Markus Bader
markus.bader@tuwien.ac.at
TU Vienna, Austria

Tutorial and Workshop Chair

Ben Brosgol
brosgol@adacore.com
AdaCore

Industrial Chair

Jacob Sparre Andersen
jacob@jacob-sparre.dk
JSA Research & innovation, Denmark

Exhibition Chair

Ahlan Marriott
ahlan@Ada-Switzerland.ch
White Elephant GmbH, Switzerland

Publicity Chair

Dirk Craeynest
Dirk.Craeynest@cs.kuleuven.be
Ada-Belgium & KU Leuven, Belgium

Local Chair

Markus Bader
markus.bader@tuwien.ac.at
TU Vienna, Austria

General Information

The **22nd International Conference on Reliable Software Technologies – Ada-Europe 2017** will take place in Vienna, Austria. Following its traditional style, the conference will span a full week, including a three-day technical program and vendor exhibition from Tuesday to Thursday, along with parallel tutorials and workshops on Monday and Friday.

Schedule

15 January 2017	Submission of papers, industrial presentation, tutorial and workshop proposals.
26 February 2017	Notification of acceptance to all authors
19 March 2017	Camera-ready version of papers required
30 April 2017	Industrial presentations, tutorial, and workshop material required

Topics

The conference has over the years become 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 representing industry, academia and government organizations active in the promotion and development of reliable software technologies.

This edition of Ada-Europe features a focused **Special Session on Reliable and Safe Robotics**. Following the increasing trend of robotic systems in industrial and public environments it is more and more important to address software systems to control autonomous vehicles. This special topic discusses issues regarding challenging problems in the field of autonomous navigation and sensor fusion. Topics include (but are not limited to): **Frameworks for robotics, planning and system modelling, as well as multi-agent and logistics applications**.

For the **general track of the conference**, topics of interest include but are not limited to (full list in the website): Real-Time and Embedded Systems, Mixed-Criticality Systems, Theory and Practice of High-Integrity Systems, Software Architectures, Methods and Techniques for Software Development and Maintenance, Formal Methods, Ada Language and Technologies, Software Quality, Mainstream and Emerging Applications, Experience Reports in Reliable System Development, Experiences with Ada.

Call for Regular and Special Session Papers

Authors of papers which are to undergo peer review for acceptance are invited to submit original contributions by 15 January 2017. Paper submissions shall not exceed 14 LNCS-style pages in length. Authors for both the general track and the special session shall submit their work via EasyChair at https://easychair.org/conferences/?conf=adaeurope2017. The format for submission is solely PDF.

The International Conference on Reliable Software Technologies is ranked class A in the CORE ranking and Microsoft Academic Search has it in the top third for conferences on programming languages. The conference is listed in DBLP, SCOPUS and Web of Science Conference Proceedings Citation index, among others.

Proceedings

Conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the conference. Camera-ready accepted papers are due strictly by 19 March 2017 (format guidelines available at in the conference site). Failure to comply and to register for the conference by that date will prevent the paper from appearing in the proceedings

Call for Industrial Presentations

The conference seeks industrial presentations which deliver value and insight but may not fit the selection process for regular papers. Authors are invited to submit a presentation outline of exactly 1 page in length by 15 January 2017. Submissions shall be made via EasyChair following the link https://easychair.org/conferences/?conf=adaeurope2017. The format for submission is solely PDF.

The Industrial Committee will review the submissions and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by 30 April 2017, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the *Ada User Journal* (http://www.ada-europe.org/auj/), which will host the proceedings of the Industrial Program of the Conference. For any further information please contact the Industrial Chair directly..

Awards

Ada-Europe will offer honorary awards for the best regular paper and the best presentation.

Call for Tutorials

Tutorials should address subjects that fall within the scope of the conference and may be proposed as either half- or full-day events. Proposals should include a title, an abstract, a description of the topic, a detailed outline of the presentation, a description of the presenter's lecturing expertise in general and with the proposed topic in particular, the proposed duration (half day or full day), the intended level of the tutorial (introductory, intermediate, or advanced), the recommended audience experience and background, and a statement of the reasons for attending. Proposals should be submitted by e-mail to the Tutorial Chair. The authors of accepted full-day tutorials will receive a complimentary conference registration as well as a fee for every paying participant in excess of 5; for half-day tutorials, these benefits will be accordingly halved. The *Ada User Journal* (http://www.ada-europe.org/auj/) 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 week. Workshop proposals should be submitted to the Workshop Chair. The workshop organizer shall also commit to preparing proceedings for timely publication in the *Ada User Journal* (http://www.ada-europe.org/auj/).

Call for Exhibitors

The commercial exhibition will span the three days of the main conference. 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.

Grants for Reduced Student Fees

A limited number of sponsored grants for reduced fees is expected to be available for students who would like to attend the conference or tutorials. Contact the Conference Chair for details.

Venue

The conference will take place at Palais Eschenbach (see images below), in the heart of Vienna, Austria.

Tools to get you there. Safely.

Ada and The GNAT Pro High-Integrity Family

www.adacore.com

AdaCore
The GNAT Pro Company

Special Session Summary: Ada and Parallelism

Pat Rogers
AdaCore, USA

Abstract

This paper provides a summary of the "Special Session on Ada and Parallelism" that took place at the 21st International Conference on Reliable Software Technologies – Ada-Europe 2016. The session included position papers from Tucker Taft and Brad Moore (published in this issue of the Ada User Journal), followed by an open discussion. The session was moderated by Jeff Cousins, ARG Rapporteur.

1 Summary

Tucker opened the discussion with a brief presentation on the work done by the "Gang of Four" (GoF), the small working group exploring how best to provide parallelism in Ada. He introduced the notion of "tasklets" in particular, and the notion of additional syntax to express parallelism. In particular he presented how to generalise the existent proposal to Containers [1].

A listener asked whether tasklets run in parallel and whether they are managed by the operating system. Tucker answered that they do indeed run in parallel and are managed at the user level.

A listener asked why not use a thread-pool instead of tasklets or any new form of parallelism. Several people answered that the overall idea is similar, but that tasklets scale to machines having hundreds of processors, whereas thread-pools do not. In addition, although a thread-pool allows smoothing if the work is not evenly distributed, tasklets do also, in that chunks of individual work can be assigned accordingly.

Brad then presented library-based approaches [2,3]. He showed a strict library-based approach (his Paraffin library of generic units), another approach based on both syntax and library calls using loops, and a Java 8 facility.

The Java approach uses parallel streams and pipes in a fork/join framework, with lambdas to conveniently express the intermediate operations (e.g., filters). Tucker indicated that there has been much discussion about lambdas in the ARG, but that careful use is required due to their inherently dynamic nature and the resulting danger of dangling references. Overall this approach was thought to be possible technically, but it was not clear that it is the right approach for Ada.

As an alternative, a listener mentioned that a common approach is to use GPUs with languages such as OpenCL. The reaction was that the resulting code is impenetrable, and that this use is expected (hoped) to be nothing more than a "phase."

The presenters noted that, in the future, compilers should be good enough at inserting parallelism during the source transformation that programmers will likely not give it explicit thought, any more than programmers now use the "register" hint to C compilers.

The question was again asked as to whether Ada tasking is not already sufficiently powerful to express parallelism, assuming compiler transformations. The answer was, again, that tasks do not scale up to machines having hundreds of processors, and that in some contexts (containers) the compiler cannot be expected to transform task statements into parallel accesses.

A "straw poll" was then taken between the pure syntax approach and the library-based approaches. The results were as follows:

1) Pure syntax: approximately half voted for this approach

2) The pure library approach: no votes

3) The combined syntax-API approach: some, not many, votes

4) The Java parallel streams approach: a small number of votes

Essentially everyone present voted to do "something" about parallelism.

The discussion continued with a question of whether multicore machines of the future will be able to execute heterogeneous code. The thought was that a degree of heterogeneity would be available but that homogeneity will always be present. One attendee asserted that programmers will rely on the compiler to utilize the machine well and will not focus on that aspect, but others disagreed.

The final topic discussed was potential recursion under the schemes presented. The presenters replied that the various schemes should not present a problem in that regard.

References

[1] Taft, S.T. (2016), *Ada Container Iterators for Parallelism and Map/Reduce*, Ada User Journal, Vol. 37, n.2.

[2] Moore, B. (2016), *Paraffin: a Parallelism API for Multiple Languages*, Ada User Journal, Vol. 37, n.2.

[3] Moore, B. (2016), *Parallel Reduction Lists: a Data Structure and Algorithm for Non-commutative Parallel Reduction with Bounded Storage*, Ada User Journal, Vol. 37, n.2.

Ada Container Iterators for Parallelism and Map/Reduce

S. Tucker Taft
AdaCore, Inc., USA

Abstract

This paper presents a proposal for syntax extensions for Ada to support parallel iterators over container structures.

1 Generalized Parallel Iterators

As multicore processors have become the norm, programming languages have begun to add syntactic, library, or pragma-based support for utilizing fine-grained parallelism (e.g. OpenMP [1], CPlex [2]). Many of these additions focus on allowing loops over a data structure (typically array or array-like) to be parallelized across multiple cores, often in conjunction with a *reduction operation* that is applied to some function of the elements of the data structure. This combination of applying a function to each element, and then combining the results, has been dubbed a *map/reduce* operation, where *map* relates to applying a function to each element, and *reduce* relates to the operation used to combine two results into a single result. This paper proposes syntax extensions for Ada to support generalized parallel iterators over arbitrary container structures, optionally with map/reduce functionality, as an outgrowth of an earlier proposal for parallel handling of arrays in Ada [3].

Ada 2012 [4] introduced *syntactic sugar* for iterating over *container* types that provide cursors and First/Next/Has_Element (and optionally Last/Previous/Has_Element) operations:

```
for Elem of [reverse] Container loop
   … -- make some use of Elem
end loop;
```

This expands into operations involving a cursor initialized to refer to the First (or Last) element of the Container, and then a loop which continues as long as the Has_Element function returns True, calling Next (or Previous) to advance through the elements. In an earlier paper [3], a syntax for doing parallel iteration over an array was proposed, including the idea of *chunked* iteration and parallel *reduction* arrays:

```
for I in parallel Arr'Range loop
   Partial_Result(<>) := Partial_Result(<>) + Arr(I)**2;
end loop;
Final_Result := Partial_Result'Reduced;
```

In this paper we will describe a combination of these approaches, by adding a Split operation to a container, which will initialize an array of cursors, with each cursor value identifying the start of a *chunk* of the container. In addition we will describe a generalization of the parallel reduction array to a *hyperobject,* which can be used to accumulate partial results and perform reduction over the partial results in parallel with the iteration over the container:

```
for Elem of parallel(Num_Chunks) Container loop
   Hyper_Obj(<>) := Hyper_Obj(<>) + Elem**2;
end loop;
Final_Result := Hyper_Obj.Reduced;
```

The parameter after **parallel** is optional, and indicates the maximum number of chunks into which the container should be split. In the absence of this, the compiler selects a default based on the number of CPUs available for use. This parameter is used to determine the length of the array of cursors to be initialized by the Split operation, which given the Container initializes the array with starting cursors for each chunk. Each chunk of the container is processed sequentially, starting at the corresponding element of the cursor array, using Next to advance the cursor until reaching the end of the container, or the start of the following chunk.

2 Splitting Container Iteration into Chunks

The proposed new form of parallel iteration relies on the container being able to provide multiple starting points within the iteration, so separate tasklets can concurrently process separate parts of the container. The new proposed operation is called Split, which initializes an array of cursors as starting points. To go with this new operation, there is a new interface, called a Parallel_Iterator, which inherits First and Next operations from Forward_Iterator, but also has the Split operation.

```
generic
   type Cursor_Array;
package Ada.Iterator_Interfaces.Parallel_Iterators is
   type Parallel_Iterator is limited interface and
        Forward_Iterator;

   procedure Split (Object : Parallel_Iterator;
                    Cursors:  out Cursor_Array);
end Ada.Iterator_Interfaces.Parallel_Iterators;
```

Given an iterator object that implements the Parallel_Iterator interface, rather than starting from the cursor returned by the First operation (and proceeding until Has_Element returns False), the Split operation may

be invoked on an array of cursors to get multiple starting points. Then a separate tasklet may be spawned for each element of the cursor array, to iterate from the corresponding element of the cursor array until the cursor equals the next element of the array (or No_Element, if starting from the last cursor of the array). Hence, a loop of the following form:

```
for Elem of parallel(Num_Chunks) Container loop
   Process (Elem);
end loop;
```

would expand into:

```
declare
   Iter : Parallel_Iterator'Class := Iterate(Container);
   Cursors : Cursor_Array(1..Num_Chunks);
begin
   Split (Iter, Cursors);  -- Get starting points for each
                           -- chunk
   for I in parallel Cursors'Range loop
      -- One tasklet per chunk
      declare
         Curs : Cursor := Cursors(I);
         End_Curs : constant Cursor :=
           (if I = Cursors'Last then No_Element else
                 Cursors(I+1));
      begin
         while Curs /= End_Curs loop
            --  Process the chunk sequentially
            declare
               Elem : Elem_Type renames Container (Curs);
            begin
               Process (Elem);
               Curs := Iter.Next (Curs);
            end;
         end loop;
      end;
   end loop;
end;
```

The Split operation will choose the starting cursors for each chunk by trying to distribute evenly the elements of the container between the chunks, while not incurring undue overhead in finding the intermediate cursor values. For example, given a balanced tree structure, Split might choose starting points by doing a walk of the tree down enough levels to provide the appropriate number of subtrees that cover the whole tree. Similarly, given a hash table, it might divide the hash range by the number of chunks, and start at the corresponding points in the hash table "backbone." Alternatively, the Split operation could determine the total number of elements in the iteration, and then use the Next operation the appropriate number of times to locate evenly-separated starting points. Presumably this latter approach would incur more overhead, but would guarantee even distribution of elements to chunks.

3 Supporting Map/Reduce over Containers

Map/Reduce is a highly parallelizable approach to summarizing information from a large set of elements. The "Map" part refers to computing some function of each element, while the "Reduce" part refers to combining the results of these computations into a single summary for the entire set of elements. This process is parallelized by breaking the overall set into subsets (or "chunks"), summarizing each subset in a separate thread (task or tasklet [5]), and then combining the summaries into a single summary using the reduction operation. We would like to simplify the use of the Map/Reduce approach in conjunction with the proposed chunked parallel iteration over containers.

The "Map" part of Map/Reduce is defined by essentially an arbitrary function, and this part is easily supported in Ada without additional constructs. The "Reduce" part is a bit more challenging, in that it requires having a separate "accumulator" for each independent thread to avoid data races, along with a mechanism for combining these separate accumulators using the reduction operation, preserving order in case the reduction operation is not commutative (for example, the concatenate operation is a non-commutative reduction operation).

To support Map/Reduce in the context of a chunked parallel iterator, the main syntactic addition needed is some way to get the index of the current chunk, suitable for indexing into an array or indexable container whose length is determined by the number of chunks. The simplest proposal is to use a special symbol for this, namely "<>", which when used inside a chunked parallel iterator will refer to the particular chunk being processed by a given iteration. Hence, repeating the example in the introductory section:

```
for Elem of parallel(Num_Chunks) Container loop
   Hyper_Obj(<>) := Hyper_Obj(<>) + Elem**2;
end loop;
Final_Result := Hyper_Obj.Reduced;
```

we see the use of "<>" as an index into the indexable Hyper_Obj container. From the expansion of the chunked parallel iterator example, we see the outer loop introduced by "**for** I **in** Cursors'Range **loop** …" uses the variable "I" as the chunk index, and hence the "<>" symbol would be replaced by "I" in this expansion. Of course, we cannot actually use a simple name like "I" in such an expansion, as "I" might already be in use, so it would probably end up being something like "_chunk_index_1" using a syntax for the identifier that could not collide with any user-declared entity.

As an alternative, we could allow the explicit declaration of a chunk index, and thereby avoid the use of the "<>" symbol, by borrowing the syntax used for an entry index specification of an entry body:

```
for Elem of parallel(C in 1 .. Num_Chunks) Container
loop
    Hyper_Obj(C) := Hyper_Obj(C) + Elem**2;
end loop;
Final_Result := Hyper_Obj.Reduced;
```

This approach avoids the already heavily overloaded symbol "<>" at the cost of a somewhat more verbose syntax.

The final step is to provide a container for the accumulators of partial results. In the above, we imagine a "hyper-object" container type, which provides a vector of accumulators indexable by the chunk index, and also keeps track of the reduction operation so that it can automatically perform the final reduction over the vector of partial result accumulators. This could be provided by a generic declared as follows:

```
generic
  type Element_Type is private;
  Identity : in Element_Type;
  with function Reducer (Left, Right : Element_Type)
       return Element_Type;
package Hyper_Objects is
  type Accumulator (Count : Positive) is tagged
       private with Variable_Indexing => Reference;

  procedure Update (Accums : in out Accumulator;
                    Index : Positive;
                    Next_Elem : Element_Type);
  -- Incorporate next element into accumulator with
  -- given index using Reducer function, where
  -- Next_Elem is the Right operand
  function Reduce (Accums : Accumulator) return
       Element_Type;
  -- Perform final reduction over vector of accumulators
  -- preserving left-to-right order

  type Reference_Type (Element : not null access
       Element_Type) is private
      with Implicit_Dereference => Element;

  function Reference (Accums: aliased in out
       Hyper_Object; Index : in Positive)
      return Reference_Type;
       -- Return reference to given element of vector
       -- of accumulators
end Hyper_Objects;
```

The above presumes the Element_Type is a definite subtype. We would need another generic if we wanted to permit Element_Type to be indefinite (e.g. an unconstrained array) indefinite types, such as an unconstrained array:

```
generic
  type Element_Type(<>) is private;
  Identity : in Element_Type;
  with function Reducer (Left, Right : Element_Type)
       return Element_Type;
package Indefinite_Hyper_Objects is
  type Accumulator (Count : Positive) is tagged
       private
  with Variable_Indexing => Reference;
  procedure Update (Accum  : in out Accumulator;
                    Index  : Positive;
                    Element : Element_Type);
    -- Incorporate next element into accumulator with
    -- given index using Reducer function, where
    -- Element is the Right operand
    ... -- As above for definite Element_Type
end Indefinite_Hyper_Objects;
```

This generic package would be useful for a hyper-object of, for example, Strings. The Update operation would be more useful in this case than directly referencing a single element of the accumulator vector, because the Update procedure would take care of re-allocating space for the result of reduction, as the accumulated result might change in size after each reduction. Here is an example of such a use:

```
declare
   package String_Hyp_Objs is new
    Indefinite_Hyper_Objects (String, Identity => "",
       Reducer => "&");
   String_Accum : String_Hyp_Objs.Accumulator
        (Count => Num_Chunks);
begin
  for Elem of parallel(C in 1 .. String_Accum.Count)
       My_Str_Vector loop
    String_Accum.Update (Index => C,
                         Element => Elem);
    -- Concatenate Elem onto end of growing
    -- accumulated result
  end loop;
  Put_Line (String_Accum.Reduce);
  -- Put concatenation of all strings from My_Str_Vector
end;
```

There is nothing special about these "hyper-object" generic packages, in that they could be implemented directly in Ada by a user. Nevertheless, there would probably be an advantage to providing something like these "definite" and "indefinite" generic packages as standard containers, to give a basic starting point for users doing these sorts of parallel Map/Reduce operations.

Conclusions

We have shown that generalizing the past proposals for chunked iteration over arrays to also support containers is relatively straightforward, with the addition of a Split operation to provide multiple starting points within a container for concurrent iterators. We have also shown that providing a way to refer to the chunk index allows for parallelized Map/Reduce over containers. Hence, we would recommend that when support for parallel iteration and Map/Reduce is added to Ada, the constructs should be general enough to support the parallel iteration and reduction of data stored in containers as well.

References

[1] OpenMP Architecture Review Board (2015), *OpenMP Application Program Interface*, Version 4.5, available at http://www.openmp.org/mp-documents/openmp-4.5.pdf, last accessed January 2016.

[2] *Programming languages — C — Extensions for parallel programming*, N1966 (2015-09-14), available at http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1966.pdf, last accessed January 2016.

[3] International Standards Organization (2012), *ISO IEC 8652:2012. Programming Languages and their Environments – Programming Language Ada.*, Geneva, Switzerland.

[4] S. T. Taft, B. Moore, L. M. Pinho, S. Michell (2014), *Safe Parallel Programming in Ada with Language Extensions*, Proceedings of the 2014 ACM SIGAda annual conference on High integrity language technology, ACM, NY, USA. http://dx.doi.org/10.1145/2663171.2663181.

[5] L. M. Pinho, B. Moore, S. Michell, S. T. Taft (2015), *An Execution Model for Fine-Grained Parallelism in Ada*, Proceedings of the 20th Ada-Europe International Conference on Reliable Software Technologies, Madrid Spain. http://dx.doi.org/10.1007/978-3-319-19584-1_13.

Paraffin: a Parallelism API for Multiple Languages

Including Ada, C, C++, C#, Java, FORTRAN, Python, Rust

Brad Moore

General Dynamics, Canada, brad.moore@gdcanada.com

Abstract

This paper presents the Paraffin parallelism API that is callable from various languages including Ada, C, C++, C#, Java, FORTRAN, Python, and Rust. One of the challenging areas that parallel frameworks must consider is parallel loop reductions, which has led to a diverse set of solutions across various languages and frameworks, and to that end, this paper focuses on problems associated with loop reduction. The existing parallelism solutions tend to require knowledge of the data types associated with the reduction results, and thus tend to be language-specific or framework-specific solutions. The simplicity of the API allows for interoperability with languages such as C, which in turn allows for interoperability with languages that can inter-operate with C. None of the other parallelism frameworks surveyed were found to provide this level of inter-language interoperability. Further, the libraries are implemented in Ada, which can be compiled to Java byte code, and CIL (.Net) byte-code, so the portability of the library is extended to frameworks associated with those environments. Although special syntax is not needed to issue the calls to the library, this paper considers how some possible syntax improvements to Ada could further facilitate making such calls in a manner more closely resembling the simple sequential loop solution to the problem.

Finally, this paper presents some performance results comparing variants of the Paraffin API calls, including those that map to implementations of OpenMP and Cilk.

1 Introduction

The need for parallel programming is becoming more common due to the increasing prevalence of multicore processors. Most common programming languages did not initially have support for concurrency and parallelism, and while Ada [1] has had such support from the beginning, it is currently lacking standardized support for fine grained parallelism [2 - 5]. With the advent of multicore processors, languages are looking at standardizing parallelism features [6], and language extensions and parallelism frameworks have become available such as OpenMP [7] and Cilk [8]. In addition, library based approaches are available such as TBB [9], and Paraffin [10].

The advantage of a syntax based approach is that it potentially allows an optimal ease of expression for the programmer, at the expense of adding complexity to the language definition. Conversely, the advantage of a library based approach is that it can typically be easier to implement for compiler writers, and easier to add to the language standard, without adding any burden to the language complexity both for the standard and from the programmers' perspective, at the possible expense of ease of expression.

The goal of any framework is that it be easy to use and understand for the programmer, while minimizing complexity to the language definition, hence it is a trade-off that must be weighed.

Arguably, the aspect of parallel programming where it is the most difficult to satisfy these goals is in the area of parallel reductions. One only has to look at the variety of creative reduction capabilities in the existing frameworks, to get a sense of the complexity of the problem space. A desire for any framework is adoption, and to that end, it is desirable if the framework can be made available in different languages. OpenMP is an example of such a framework, since it provides parallelism support for C, C++, and FORTRAN, but it is a syntax based solution, since it requires compiler support to provide the necessary transformation to apply the parallelism pragmas to the user's code. Extending OpenMP to support other languages is a possibility, but is a non-trivial exercise since adding language extensions to a new language involves compiler support, not to mention extending the OpenMP standard itself to support the new language, and the resulting extensions typically are considered to be outside the language definition, and therefore non-portable. A much easier path to a framework that has inter-language interoperability is possible if the framework is a library based approach, rather than a syntax based approach. Most library based solutions (e.g. TBB [9], TPL [11], Java Streams [12]) however, are still framework specific because they use language specific syntax such as generics or templates to allow the library to compute the results for parallel reduction. Object oriented features such as class hierarchies also tend to be non-portable between languages, or at least make it difficult for multi-language use, since not all languages support object oriented constructs, and those that do may have unique, non-compatible approaches that limit cross language portability.

This paper presents part of the Paraffin API, which is a library based API written in Ada that is designed to be callable from multiple languages including; Ada, C, C++, C#, Java, Python, FORTRAN, and Rust.

The multi-language Paraffin API library calls do not involve generics, template, or object oriented constructs, and the reduction capabilities are completely decoupled from the user data types associated with the reductions. All reduction code involving user reduction data types is managed by the client of the library. None of these library calls accept parameters that designate reduction results. This allows the library calls to be easily adaptable to being called from multiple languages. The API is also easily adaptable to providing an alternate interface to implementations of other parallelism frameworks such as the GOMP OpenMP ABI and the gcc Cilk ABI, allowing a common API to be applied to implementations of these other frameworks.

Java and C# are also options since there exists Ada compilers that can generate CIL byte code for C#, or Java byte code for Java. The advantage is that since it is a library based approach, no special compiler support is needed.

The paper then goes on to describe how a library based approach might be optionally enhanced for usability with syntactic sugar, particularly in Ada to further facilitate making such library calls. The overall idea is to explore alternatives that might either complement or reduce the need to add specialized parallelism syntax to the Ada language. Ultimately it may make sense for Ada to add special syntax support for fine-grained parallelism, but a library approach may be of interest to those who need specific behaviour or implementation transparency, that might not be associated with any eventual syntax. For instance, features of Paraffin such as no heap allocation, no queueing of work items, bounded storage for reductions, stack-safe parallel recursion, might be desired. A library that can be called from other languages might also be of interest to users of those other languages, particularly if parallelism support is not already available in those languages.

The areas of syntactic sugar improvements of interest include some form of anonymous subprogram calls, and usability improvements for thread local storage. Anonymous subprograms would be a general feature that could be applied to libraries with callbacks, not necessarily restricted to parallelism calls. A secondary proposal is presented to suggest a mechanism for making it easier to use thread local storage in Ada. Currently, Ada.Task_Attributes can be used for this purpose, but requires generic instantiation, which is not as convenient as mechanisms that can be found in other languages.

2 Infrastructure for a Common Parallel Loop Reduction API

As stated above, in order for a library to be callable from a wider variety of programming languages, it must not present an API that utilizes language features that are not commonly available. For instance, this rules out the use of templates, generics, concurrency related features, and object oriented constructs in the definition of the API call, although note that it does not necessarily rule out the use of these features in the user supplied callbacks associated with the framework, since the user supplied callbacks are unrestricted in the domain of the client. Ideally, the API should not be required to interact with or be aware of any user defined data types.

It turns out that a relatively simple API is possible, but it does require the support of a small set of language capabilities. Either the language needs to be interoperable with Java byte code, or dot net CIL byte code, or the language as a bare minimum needs to support;

- Calling C functions,
- Passing native language callbacks as parameters to C functions,
- Calling back into the native language from C from external threads

Ideally, the language would support some form of thread local storage (TLS), but this is not strictly necessary, since the Paraffin libraries expose library routines that provide this capability, that are callable from C.

Ideally also, the language would provide support for nested closures such as nested subprograms in Ada, or lambdas in C++. This allows callback routines to access variables in enclosing but nested scopes, which typically is needed for assignment of the final reduction results. This feature also is not strictly necessary, because there are forms of the library calls that accept a context parameter, which can be used to pass references to variables of nested scope, which is a technique commonly used in C.

3 Thread Local Storage (TLS)

Thread local storage provides a convenient mechanism for the client to declare partial result variables that can be accessed safely by parallel executors or worker threads, without the need for locking or synchronization between threads. A benefit of thread local storage variables is that the parallelism libraries do not necessarily need to be aware of these user defined partial result variables, if these variables are only referenced in user specified callbacks. At some point, these partial result variables may need to be reduced into a single result variable stored in a global scope, but if the reduction is also specified as a user supplied callback, then the reduction can safely be used to coordinate reductions between multiple threads of execution, as will be shown below.

3.1 Thread Local Storage in C

The keyword ThreadLocal was added in C11 [13] to specify thread local storage for variable declarations. The header, <threads.h>, if supported, defines thread_local as a synonym for that keyword.

```
#include <threads.h>
thread_local int partial_sum = 0;
```

3.2 Thread Local Storage in C++

In C++11, the thread_local keyword was added for Namespace level (global) variables, file static variables, function static variables, and static member variables.

3.3 Thread Local Storage in C#

In C#, there are at least 3 options; one can apply the ThreadStatic attribute to a static field. In .Net 4.0 or above, one can use the System.Threading.ThreadLocal<T> class which also provides lazy initialization, and is not limited to static fields, or finally one can use Named Data Slots capability if the other two options are not applicable.

e.g.,

```
[ThreadStatic]
private static int partial_sum;
```

or alternatively using the ThreadLocal class;

```
ThreadLocal<int> PartialSum =
 new ThreadLocal<int>(() => {return 0;});
```

In this particular case, the constructor is using a valueFactory anonymous lambda function to initialize the thread local variable in each thread.

3.4 Thread Local Storage in Java

Java similarly provides a ThreadLocal class, where the initial value can be specified by type derivation and overriding the initialValue() member function. e.g.,

```
private static final ThreadLocal<Integer> PartialSum =
    new ThreadLocal<Integer>() {
     @Override protected Integer initialValue() {
       return 0;}
```

3.5 Thread Local Storage in Ada

Since Ada 95, Ada [1] has provided thread local storage in the form of the Ada.Task_Attributes standard library. Similar to Java and C#, this library is a generic package that requires instantiation. e.g..

```
package Integer_Attributes is new
  Ada.Task_Attributes
    (Attribute => Integer, Initial_Value => 0);

Partial_Sum : Integer := Integer_Attributes.Value;
```

In addition, the GNAT compiler provides the non-standard pragma, available on most common platforms called **Thread_Local_Storage**, which may be applied to variable declarations in library level packages. The pragma is more efficient than the Task_Attributes package, although for the purposes of this common parallelism API, the task_attributes can be used in a manner such that its less efficient implementation does not significantly affect the parallelism performance results. The pragma however is useful in other situations when interfacing with foreign threads, for example.

3.6 Thread Local Storage in Other Languages

Not all languages support thread local storage. For those that do not (e.g. FORTRAN, Python, Rust), Paraffin provides library calls for elementary types such as 32 and 64 bit integers, and 64 and 128 bit floating point values.

Multiple values of any of these types may be stored in thread local storage, as each value can be associated with a unique numeric id lookup value.

For example, for the C long integer type, the following package exists.

```
package Parallel.Task_Local.Long is
  ...

  function Get_Long
   (Id : Integer) return C.long
   with Export,
      Convention => C,
      External_Name =>
        "Paraffin_Task_Local_Get_Long";

  procedure Set_Long
   (Id  : Integer;
    Val : C.long)
   with Export,
      Convention => C,
      External_Name =>
        "Paraffin_Task_Local_Set_Long";

end Parallel.Task_Local.Long;
```

4 Possible future improvements for Ada with TLS

The Ada standard package, Ada.Task_Attributes turns out to be useful for parallelism, since it allows one to place the storage for variables within a task. For parallelism however, use of this package with parallel executors implies that the executors are implemented as Ada tasks, which may not be the case.

The GNAT pragma Thread_Local_Storage might be better suited for this purpose, because it can work with foreign threads, but still there is the implication that OS threads are being used, which might not necessarily be the case.

Furthermore, currently in Ada, there are significant restrictions on where the Thread_Local_Storage pragma can be placed, that are not as restricted in other languages. It would be useful if the pragma could be placed on variable declarations declared within nested scopes. The idea being that the visibility of the variable is limited to the same scope of visibility as other variables declared at the same site.

Taking this idea one step further, it would be nice if the thread local storage associated with a partial results variable of a parallelism computation could be more closely coupled to the final result variable. Rather than declare two separate variables, an aspect could be defined, such as Parallel, which could be applied to a variable declaration. A new attribute 'Partial could also be defined and applied to a parallel variable to distinguish between thread local usage and the final result instance.

```
declare
   Sum : Natural with Parallel := 0;
begin
  ...
end;
```

The intent of the declaration of Sum above is that there is a regular local variable called Sum, as well as "shadowing" independent thread local versions of variables with the same properties and initialization. To reference the local variable, it can be named as per existing Ada syntax. To reference the thread local version, one can use the 'Partial attribute such as:

```
Sum'Partial := 10; -- Update the TLS Sum variable
```

5 Nested callback scope

When reductions are needed, it is quite typical that the reduction result is declared in an enclosing but nested scope.

E.g.,

```
declare
   Sum : Integer := 0;
begin
   for I in 1 .. 1000 loop
      Sum := Sum + Arr(I);
   end loop;
end;
```

For example, one could parallelise the above loop, by replacing the loop with a call to a library subprogram, where the library call breaks the loop iterations into chunks allowing different executors to process different sets of iterations of the loop in parallel. The Sum variable cannot be safely updated by the executors however, as that would be a data race, so typically each executor operates on a thread local copy of the sum variable, and then during reduction, as each chunk completes, or after all chunks are complete, the partial results are combined into a single result and stored in the Sum variable. The issue is that if the library call involves a user supplied callback to execute the loop, it presents a problem if the language in question does not allow nested callbacks, because otherwise the callback subprogram must then be declared at library level, and would not have visibility to the nested Sum variable declaration, which would need to be referenced in the callback. Alternatively, the Sum variable could be moved to a library level declaration, but that is generally undesirable, since it breaks up the flow of the program which affects readability, and limits the potential to use the stack, which also can introduce data races if more than one thread of execution needs to execute the loop concurrently. Here we examine the nested callback capability across several languages.

5.1 Nested callbacks in C

Currently the C standard does not support nested function calls, nor is there a standard mechanism to access nested global scope from within a callback function. One can however, work around these limitations using a context parameter, which is a void pointer that can be mapped to a user defined data type.

```
void Callback (void *context)
{
   int i;
   int *sum = (int *) context;
   for (i = 0; i < 1000; i++)
      *sum += Arr[i];
}

int main()
{
  int sum = 0;
  CallCallback(&sum, &Callback);
  /* Sum address is passed as context */
}
```

The gcc C implementation does support nested subprograms as a non-standard language extension, which would allow one to write;

```
int main()
{
  int sum = 0;

  void Callback()
  {
     int i;
     for (i = 0; i < 1000; i++)
        sum += Arr[i]; /* Sum is visible */
  }

  CallCallback(&Callback);
}
```

The Paraffin libraries support both of these mechanisms for C interoperability.

5.2 Nested callbacks in C++

None of the C based languages (C++, C#, Java) support nested functions in their respective standards, and only the gcc C implementation provides non-standard nested function support. The other languages have various forms of lambda function capabilities or anonymous classes however, which provides a similar capability to allow referencing variables from enclosing nested scopes, while also allowing the callback to be written inline, which is a more natural flow for the reader, and eliminates clutter by eliminating the name associated with the callback function. In C++, one can write;

```
{
  int sum = 0;
  callCallback([&sum]
   {
     for (int i=0; i < 1000; i++) {
        sum += arr[i];
     }
   });
}
```

Note that the sum variable is mentioned at the start of the lambda function enclosed in square brackets. This is called a capture, which means that the variable from the enclosing scope may be referenced from inside the lambda function. The ampersand indicates that the variable is captured by reference, which is needed if we want the sum variable from the enclosing scope to be updated.

5.3 Nested callbacks in C#

```
int sum=0;
callCallback (callback: () =>
    {
    for (int i = 0; i < 1000; i++) {
      sum += arr[i];
    }
  });
```

The C# solution is very similar to the initial C++ solution, except that all variables of the enclosing scope are automatically captured and thus do not need to be mentioned in capture syntax.

5.4 Nested callbacks in Java

While Java does have lambda expressions, these are functions that return the result of an expression, which is not what is needed for the parallelism API. For callbacks, the closest equivalent of the lambda in C# or C++ is the anonymous class.

```
private static int sum=0;
...
callCallback
    (new callback()
      {
        public void Invoke ()
          {
          for (int i = 0; i < 1000; i++) {
            sum += arr[i];
          }
      });
```

In this scenario, the callCallback function issues a call to the Invoke method.

5.5 Nested callbacks in Ada

Ada has always supported nested subprograms, which allows callbacks to be written that can access nested scope.

```
declare
   Sum : Integer := 0;

  procedure Callback is
  begin
    for I in 1 .. 1000 loop
      Sum := Sum + Arr (I);
    end loop;
  end Callback;
begin
  Call_Callback (Callback => Callback'Access);
end;
```

6 A possible future improvement for Ada – Anonymous subprograms

While, the nested subprogram capability of Ada makes it relatively easy to create callbacks that can reference variables from an enclosing nested scope, it may be worth considering whether it makes sense to add some form of syntactic sugar to allow callbacks to be written inline as an anonymous subprogram, similar to features that now exist in many other languages. It can be argued that being able to write subprograms inline improves readability by placing the logic of the callback more closer to the natural flow of the processing, and having to create names for subprograms and their parameter profiles that are only called once adds unnecessary clutter to the code. The physical separation from the call site and the callback site also interferes with the readability for linkage of the call parameters. An additional benefit is that it eliminates a need for 'Access, and having to work with access type pointers.

A possible syntax could be modelled after the approach taken by the other languages, since the end result could look very similar. Furthermore, implementation of the feature should be eased since such calls could be transformed to nested subprogram calls which exist in Ada today. With such a capability, the above Ada example might be rewritten as;

```
declare
   Sum : Integer := 0;
begin
  Call_Callback (
       <for I in 1 .. 1000 loop
          Sum := Sum + Arr (I);
       end loop>);
end;
```

The idea here is that procedural callback code is enclosed by angle brackets, and functional expression callbacks are enclosed by round brackets, similar to the syntax for Ada 2012 expression functions. For procedural callbacks, the angle brackets syntax could be constrained to allow the syntax to only directly enclose a single statement. If more than one statement is needed, a group of statements could be enclosed by a block statement inside the angle brackets.

In a manner similar to C++, C#, and Java, any parameters needed for the call could also be supplied inline, enclosed by parenthesis, as per the existing Ada syntax, except that the types and parameter modes could optionally be eliminated, since they could be inferred by the call [14]. Although other modern languages have added similar anonymous subprogram capabilities, it does not necessarily mean that the idea is a good fit for Ada.

The possibility of having statements embedded at a call site might be seen as too much of a departure from the language, particularly if the feature provides no new capabilities other than readability enhancements. Whether such features ever make it into the language standard remains to be seen, but in the meantime, Ada's nested subprogram capabilities work well with the Paraffin

libraries. The Ada examples in this paper will however, use this proposed syntax, because it makes the examples more concise and readable.

7 A Parallelism API callable from C, C++, C#, Java, and Ada

Now that some useful infrastructure has been described, in particular thread local storage, and callbacks that can access enclosing nested scope, a multi-language API that can be presented to perform parallel reductions associated with parallel loops;

```
type Iteration_Index_Type is System.Min_Int ..
System.Max_Int;

procedure Parallel
  (From, To : Iteration_Index_Type;
   Context  : System.Address;
   Reset    : not null access procedure;
   Process : not null access
     procedure (Start, Finish : Iteration_Index_Type);
   Reduce   : access
     procedure (Context      : System.Address;
                Start, Finish : Iteration_Index_Type
                  with Loop_Cursor));
```

For compilations to native languages such as C, C++, and Ada, the subprograms are exported with the C calling convention. To use this interface, the programmer declares any partial result variables external to this call, using any form of thread local storage supported by the calling language.

7.1 Terminology: Chunks and Grains

Some basic terminology is needed prior to discussing this API call. Using a divide and conquer approach, the parallel processing of a loop can be broken down into two units of work items: Chunks and Grains.

A Grain of processing is the smallest division of work. Each grain consists of a subset of iterations of the loop, where all iterations of the grain are executed by the same executor. A grain cannot be subdivided into smaller units of work. For load balancing strategies such as Work Stealing [15] or Work Seeking [16], grains are the entities that can be stolen from one executor and reassigned to an idle executor. Only whole grains can be stolen, so once an executor has started to execute a grain, it cannot be stolen by another executor.

Chunks represent a consecutive sequence of grains that are processed by a single executor. A chunk is essentially an execution of grains, and the notion of a chunk is dynamic and determined during the execution of the loop. If an idle executor steals work from another executor, it typically splits the chunk or remaining work of the busy executor into two smaller chunks, and claims one of the chunks for itself, the idle executor, which then becomes busy processing the new chunk, while the original busy executor continues to process what remains of its original chunk.

The distinction between chunks and grains disappears for static work division, called Work Sharing with respect to the Paraffin libraries. For this case, there is no dynamic load balancing, so the original chunks never get split apart, and can also be considered to be grains.

7.2 Dissecting the Parallel_Loop Call

The purpose of the Reset callback of the Parallel_Loop API call presented above is to initialize and/or reset any such thread local variables to an initial state. Executors call this callback whenever a new chunk of work is assigned to an idle executor. Each executor can process multiple independent chunks, but before the processing for each chunk is started, any thread local storage needs to be reset to the initial state again, as the thread local partial results for each chunk need to be isolated from the partial results calculated for other chunks. Since the Reset callback is intended to only affect thread local storage, it can be called by multiple executors concurrently, without any need for synchronization.

The purpose of the Process callback is to perform the main processing of the loop. Usually this involves only access to local variables and the thread local storage associated with partial results. Otherwise, global variables may be accessed, but then care must be taken to ensure that the access to such variables is properly synchronized through locks or protected objects, if the variables are updated from within the loop. The Start and Finish parameters of the Process callback identify a “grain” of iterations of the loop. The Loop_Cursor aspect is intended to allow for special proposed syntax that makes a library call to a subprogram look more like a loop. If the call is written as a procedure call however, then the aspect has no effect. For the time being, the aspect can be ignored, as it will be described and used in an example later below.

The purpose of the Reduce callback is to combine the partial results in thread local storage for a particular executors processing of a single chunk, with the final result. This typically involves applying some operation to combine the thread local storage variables with the final result variables.

Reductions are best applied once per chunk, rather than once per grain. One reason for this, is that it reduces the amount of synchronization needed to combine the partial results in the final result, which can result in performance gains.

The Reduce callback is called by the executor that stored the partial result in thread local storage, so the callback has access to the thread local storage of that executor. The Reduce callback is called from a protected context, which ensures that only one executor at a time can call Reduce. This eliminates the need for any locking in the programmer’s callback code, and global variables such as the final result variables declared in an enclosing scope can be updated and modified safely. For this callback, the Start and Finish parameters here identify the “chunk” of consecutive iterations that were processed by an Executor rather than a grain as per the Progress callback. The Reduce

callback is called before the idle Executor attempts to acquire more work (e.g. by Work Stealing, or Work Seeking). For most reduction problems these parameters can be ignored as they are not used, but certain problem can make use of these parameters, such as for storing partial results in a sorted container, sorted by loop index.

The context parameter exists for languages that do not support nested subprograms. For languages that support nested subprograms, such as Ada, and the gcc version of the C compiler, variants of the library calls exist where the context parameters have been eliminated. For languages such as standard C, the context parameter of the Parallel_Loop call can be passed the address of a final result variable from the enclosing scope. This address is in turn passed back to the Reduce callback, which can be typecasted to designate the final result variable which it originally referenced. This allows the Reduce callback to update the final result variable which otherwise would not have been visible to a callback routine that is declared at library level. If more than one reduction needs to be generated by the loop, then the address of a composite record structure can be passed instead that contains the multiple reduction values.

As an example of usage for the API, reconsider the problem of calculating the sum of an array of integers. Here we show the example using the 'Partial attribute for shadowing local variables with thread local storage, described above. The example also utilizes anonymous subprogram syntax presented above, mostly for brevity, to illustrate how parameters to callbacks could work with the syntax. Note that the example could be rewritten to use Ada.Task_Attributes and nested subprograms instead which are features available in Ada today.

```
Sum : Integer with Parallel := 0;

Parallel -- Loop
  (From    => 1,
   To      => 1000,
   Reset   => <Sum'Partial := 0>,
   Process => (Start, Finish)
     <for I in Start .. Finish loop
        Sum'Partial := Sum'Partial + Arr (I);
      end loop>,
   Reduce  => <Sum := Sum + Sum'Partial>);
```

or alternatively we can consider a syntax alternative that strives to make the call to a library look more like a loop. The first variant of this syntax could allow the parameters of the loop Process callback to be mentioned where the loop cursor iterator would normally be placed. This parameter list would be enclosed in parenthesis and could be referenced from inside the loop. The end result is a nested loop structure, which conceptually makes sense since the outer loop can be thought of as iterating through chunks, while the inner loop is iterating through a specific chunk.

```
Sum : Integer with Parallel := 0;

for (Start, Finish) of Parallel (From   => 1, To => 1000,
                    Reset  => <Sum'Partial := 0>,
                    Reduce => <Sum :=
       Sum+Sum'Partial> loop
  for I in Start .. Finish loop
    Sum'Partial := Sum'Partial + Arr (I);
  end loop;
end loop;
```

This is somewhat satisfying but what we would really like to see here is a single non-nested loop. To get there, another loop syntax variant is proposed which would involve the Loop_Cursor aspect.

A callback that has parameters identified with the Loop_Cursor aspect is allowed to be written as the body of a loop, but the inner loop is implicitly written using the Loop_Cursor parameters. Instead of a list of callback parameters enclosed by parentheses, the programmer would write a single loop cursor name without parentheses. This form would be accepted by the compiler only if all parameters of the callback can be implicitly converted by the compiler to either reduction variable references or loop cursor parameters.

In this situation, the programmer is indicating that the inner loop should be implicitly generated. The current value of the loop cursor is named by the loop cursor of the loop as per existing for-loop syntax. This could allow for rewriting the above library call to more resemble a single loop rather than a nested loop, which would be mapped to the same library subprogram call.

This could then be written as:

```
Sum : Integer with Parallel := 0;

for I of Parallel (From   => 1, To => 10_000_000,
                   Reset  => <Sum'Partial := 0>,
                   Reduce => <Sum := Sum + Sum'Partial>
loop
  Sum'Partial := Sum'Partial + Arr (I);
end loop;
```

Using this syntax increases the elegance of the solution, as it brings the code back to look more like the original version, which was a sequential for loop. The code is concise and readable, yet gives the reader an understanding that parallelism is involved, as well as an idea of how the data races are avoided and how the parallelism takes place.

7.3 Paraffin Reductions from standard C

The following example illustrates how the previous example might look, when called from C.

```
#include “paraffin.h”

static __thread int partialSum = 0;
void reset(int start) {partialSum = 0;}
void reduction(void *context, int start, int finish)
{
  int *sum = (int *) context;
  *sum += partialSum;
```

```
}

void compute(int start, int finish)
{
  int i;
  for (i = start; i <= finish; i++)
    partialSum += i;
}

void main (){
   int sum = 0;
   Parallel_Loop(1, 1000, &sum, &reset, &compute,
                 &reduction);
}
```

7.4 Paraffin Reductions from C (gcc with nested functions)

The following example illustrates how the previous example might look, when called from C, using gcc's non-standard nested functions.

```
#include “paraffin.h”
{
   static __thread int partialSum = 0;
   int sum = 0;
   void reset(int start) {partialSum = 0;}
   void reduction() {sum += partialSum;}
   void compute(int start, int finish)
   {
     int i;
     for (i = start; i <= finish; i++)
       partialSum += i;
   }
   Parallel_Loop (1, 200000000, &reset, &compute,
                  &reduction);
}
```

7.5 Paraffin Reductions from C++

```
#include <functional>
#include "paraffin.h"

static __thread int partialSum = 0;

void resetPartial (int start) {partialSum = 0;}
void computeSum (int start, int finish)
{
  for (int i = start; i <= finish; i++)
    partialSum += i;
}

{
  int sum = 0;
  auto la_reduce = [&sum](void) { sum += partialSum; };
  static std::function< void( void ) > static_reduce;
  static_reduce = la_reduce;
  void (*reduce)( void ) = [] { static_reduce(); };

  Parallel_Loop (1, 200000000,
            &resetPartial, &computeSum, reduce);
}
```

Alternatively, the standard C version above could be used, without having to use lambda functions.

7.6 Paraffin Reductions from C#

```
class test_paraffin_lib
{
   [ThreadStatic]
   private static int partial_sum;

   static void Main(string[] args)
   {
      int sum = 0;

      paraffin_pkg.Parallel_Loop
        (from: 1, to: 1000,
         reset: new Paraffin.partial_results_reset_callback
                  ((int start) => { partial_sum = 0; }),
         process: new Paraffin.Parallel_Loop_Callback
              ((int start, int finish) =>
          {
             for (int i = start; i <= finish; i++) {
                partial_sum += i;
             }
          }),
         reduce: new paraffin.reduction_callback(() =>
          { sum += partial_sum; }));
   }
}
```

7.7 Non-Commutative Reductions

Most reductions operations such as integer addition, min, max, etc, are both associative and commutative. However, some operations, such as concatenation and chain multiplication of matrices are only associative. This can lead to incorrect results if partial results are combined in an order that differs from the sequential ordering.

The multi-language API allows the programmer to specify all the details of the reduction, and otherwise only provides the necessary synchronization to prevent data races. As such the notion of non-commutative reduction is not directly supported by the API, but can be provided relatively easy by the programmer, by using techniques such as reducing to a sorted container type such as an ordered set or ordered map container. Consider the problem of concatenating all the letters of the alphabet in a parallel loop. The following shows how the API can be used to generate the correct results, while processing the loop in parallel. Note, while the multi-language Paraffin API does not directly manage non-commutative reduction except by user control as in this example, there are Ada specific generic Paraffin API calls that directly manage non-commutative reduction within the library call without having the programmer provide this management.

```
package Maps is new Containers.Ordered_Maps
(Key_Type     => Natural,
 Element_Type =>
Strings.Unbounded.Unbounded_String);

Map : Maps.Map;
```

```
Alphabet : Strings.Unbounded.Unbounded_String
      with Parallel;

for Letter in Parallel
   (From    => Iteration_Index_Type (Character'Pos('A')),
    To      => Iteration_Index_Type (Character'Pos ('Z')),
    Reset   => <Alphabet'Partial :=
     Strings.Unbounded.Null_Unbounded_String>,
     Reduce  => (Start, Finish : Iteration_Index_Type)
       <Map.Include (Key  => Start,
                     New_Item => Alphabet'Partial)>)
loop
 Strings.Unbounded.Append(Alphabet'Partial,
                  Character'Val(Letter));
end loop;
```

One could now iterate through the ordered list to generate a correct alphabet string result.

This particular problem may be better suited for one of the generic, Ada-specific Paraffin calls, which does manage non-commutative reduction.

```
generic
  type Loop_Index is range <>;
  type Result_Type is private;
  with function Reducer
    (Left, Right : Result_Type) return Result_Type;
  Identity_Value : Result_Type;
package Parallel.Generic_Loops is
  function Parallel
   (From      : Loop_Index := Loop_Index'First;
    To        : Loop_Index := Loop_Index'Last;
    Process   : not null access
      procedure (From, To : Loop_Index
                          with Loop_Cursor;
                 Result   : in out Result_Type
                          with Partial);
   ) return Result_Type;
end Parallel.Generic_Loops;
```

An advantage of using a generic call is that the result type and loop index type can exactly match the programmers data types. Another difference is that thread local storage is not needed. The partial result is passed as a callback parameter. Also, the reduction works differently. The reduction is specified as a function result of combining two partial result values. Here, the Loop_Cursor works as described above, but this introduces a new Aspect, Partial. The prefix of 'Partial attribute reference must either match a parallel variable with thread local storage as per the previous examples, or alternatively in this case, it could designate a callback parameter that has the partial aspect. If the library call is a function, then the prefix of the 'Partial attribute also denotes the name of the final result variable where the function result is implicitly assigned. This should allow us to rewrite the alphabet problem as follows:

```
package Letter_Loops is new
  Parallel.Generic_Loops
   (Loop_Index => Character,
    Result_Type =>
      Strings.Unbounded.Unbounded_String,
    Reducer => "&",
    Identity_Value =>
      Strings.Unbounded.Null_Unbounded_String);

use Letter_Loops;

Alphabet : Strings.Unbounded.Unbounded_String;

for Letter of Parallel (From => 'A', To => 'Z') loop
 Strings.Unbounded.Append(Alphabet'Partial, Letter);
end loop;
```

Note that this version does not need a temporary ordered map container to store the intermediate results, yet results in a concise loop that closely approximates the sequential version of the loop. Because the call involves Ada generics, it unfortunately cannot be called from other languages easily, but for use in Ada, it is well suited for the problem.

Using the same generic, the sum of an integer array problem could be expressed as:

```
package Natural_Loops is new
  Parallel.Generic_Loops (Loop_Index => Natural,
                          Result_Type => Natural,
                          Reducer => "+",
                          Identity_Value => 0);
use Natural_Loops;

Sum : Natural := 0;

for I of Parallel (From => 1, To => 10_000_000) loop
 Sum'Partial := Sum'Partial + Arr(I);
end loop;
```

8 Performance Measurements

The Paraffin libraries include a series of tests for various problems. To illustrate performance differences between the multi-language (non-generic) Paraffin library calls versus the generic Ada specific library calls, some tests were performed on a Ubuntu desktop running an AMD athlon processor. Three runs were conducted, one running native executables, and the other running the dotnet version of the executables running under the Mono CLS virtual machine, and the final version running the java version of the executables running as java byte code. The example tested involves the calculation the sum of 64 bit integers from 1 to 400_000_000.

All of the tests that specify Cilk or OpenMP involve Paraffin API calls that map to the respective ABI's, but where the management of the reduction is handled by the Paraffin library rather than the OpenMP or Cilk libraries. For example, there is no use of Cilk hyperobjects for the reductions. OpenMP and Cilk just provide the executors, though the loop body processing is also performed by the respective executors of the ABI. The exception is where the example was run in pure C++ for both OpenMP and Cilk using the OpenMP pragmas and cilk_for syntax with hyper objects. Those runs are identified with the "(Pure)" tag in

the listing. All other test results are for native Paraffin libraries written entirely in Ada.

For some reason, the pure Cilk test with hyperobjects performed poorly, and was considerably worse than the sequential time. It may be that the Cilk hyperobjects in the gcc implementation are not intended to be used for this sort of problem with extremely large iteration count, but little processing per iteration. It may also be that there is some problem with the Cilk configuration, although the Cilk results using the Paraffin API were much better, and within the expected times. This suggests that there may be overhead associated with hyper objects that is accentuated by this particular problem.

From the results on the native target it appears there may be a slight performance edge in running the generic library calls over the multilanguage non-generic calls. More testing would be needed to confirm this, though the difference does not appear to be very significant. Under the Mono and Java virtual machines, we see overall that the results compare very well to running natively, however one of the tests, (the work sharing tests) performed surprisingly better than any of the others on all targets, which in turns performs slightly worse than using the language specific parallelism libraries. The reasons for this difference are unexplainable at the time of writing, but may happen to be tied to that particular test scenario. It appears that the Work Sharing logic in Paraffin is similar to how Java and C# implement their parallel loops, providing performance results far better than one would expect for 4 cores. The other approaches produce results more in line with what one would expect for having 4 cores.

```
********** Parallel Framework Test **********
  Physical Processors= 4
  Executors = 4
  OS        = LINUX
  Processor = AMD Athlon(tm) II X4 635 Processor

Integer Reduction of Sum from 1 to 400_000_000

Sequential:                Elapsed = 00:00:01.11
OpenMP Sequential (Pure)    Elapsed = 00:00:01.11
Work Sharing:              Elapsed = 00:00:00.26
OpenMP (Pure) in C         Elapsed = 00:00:00.32
OpenMP Static:             Elapsed = 00:00:00.25
OpenMP Dynamic:            Elapsed = 00:00:00.25
OpenMP Guided:             Elapsed = 00:00:00.24
Cilk Work Stealing:        Elapsed = 00:00:00.25
Cilk Work Stealing (Pure)  Elapsed = 00:00:02.94
Work Seeking:              Elapsed = 00:00:00.24
Work Stealing:             Elapsed = 00:00:00.25
Work Sharing Blocks:       Elapsed = 00:00:00.27
Work Seeking Blocks:       Elapsed = 00:00:00.25
OpenMP Static Blocks:      Elapsed = 00:00:00.27
OpenMP Dynamic Blocks:     Elapsed = 00:00:00.25
OpenMP Guided Blocks:      Elapsed = 00:00:00.24
Pooled Work Sharing:       Elapsed = 00:00:00.26
Unbounded Pooled Work Sharing:
                           Elapsed = 00:00:00.25
Ordered Unbounded Pooled Work Sharing:
                           Elapsed = 00:00:00.25
Pooled Work Seeking:       Elapsed = 00:00:00.23
Pooled Work Stealing:      Elapsed = 00:00:00.32
Pooled Work Sharing Blocks:  Elapsed = 00:00:00.30
Pooled Work Seeking Blocks:  Elapsed = 00:00:00.25
Non Generic Work Sharing:    Elapsed = 00:00:00.31
Non Generic OpenMP Static:   Elapsed = 00:00:00.30
Non Generic OpenMP Dynamic:  Elapsed = 00:00:00.29
Non Generic OpenMP Guided:   Elapsed = 00:00:00.29
Non Generic Cilk W-Stealing: Elapsed = 00:00:00.35
Non Generic Work Seeking:    Elapsed = 00:00:00.28
Non Generic Work Seeking:    Elapsed = 00:00:00.27
Non Generic Work Stealing:   Elapsed = 00:00:00.29
Non Generic Pooled Work Sharing:
                           Elapsed = 00:00:00.30
Non Generic Pooled Work Seeking:
                           Elapsed = 00:00:00.29
Non Generic Pooled Work Stealing:
                           Elapsed = 00:00:00.29


********** Parallel Framework Test **********
  Physical Processors= 4
  Executors = 4
  Target    = Java JVM
  OS        = LINUX (Compiled on Windows)
  Processor = AMD Athlon(tm) II X4 635 Processor

Integer Reduction of Sum from 1 to 400_000_000

Sequential:                Elapsed = 00:00:00.98
Work Sharing:              Elapsed = 00:00:00.28
Work Seeking:              Elapsed = 00:00:00.26
Work Stealing:             Elapsed = 00:00:00.26
Work Sharing Blocks:       Elapsed = 00:00:00.31
Work Seeking Blocks:       Elapsed = 00:00:00.20
Non Generic Work Sharing:    Elapsed = 00:00:00.10
Non Generic Work Seeking:    Elapsed = 00:00:00.31
Non Generic Work Seeking:    Elapsed = 00:00:00.32

********** Parallel Framework Test **********
  Physical Processors= 4
  Executors = 4
  Target    = CIL (dotnet) byte code
  OS        = Linux (Compiled on Windows)
  Processor = AMD Athlon(tm) II X4 635 Processor

Integer Reduction of Sum from 1 to 400_000_000


Sequential (Ada):          Elapsed = 00:00:00.98
Sequential (C#):           Elapsed = 00:00:00.93
C# Foreach loop:           Elapsed = 00:00:00.09
Work Sharing:              Elapsed = 00:00:00.28
Ordered Work Sharing:      Elapsed = 00:00:00.27
Work Seeking:              Elapsed = 00:00:00.26
Ordered Work Seeking:      Elapsed = 00:00:00.27
Work Stealing:             Elapsed = 00:00:00.27
Ordered Work Stealing:     Elapsed = 00:00:00.28
Work Sharing Parallel Blocks:Elapsed = 00:00:00.32
Work Seeking Parallel Blocks:Elapsed = 00:00:00.20
Non Generic Work Sharing:    Elapsed = 00:00:00.12
Non Generic Work Seeking:    Elapsed = 00:00:00.39
Non Generic Work Stealing:   Elapsed = 00:00:00.33
```

Conclusion

This paper presented a parallelism API that has been implemented in Paraffin, and can be called from multiple languages including C, C++, C#, Java, FORTRAN, Python, Rust, and Ada. The paper also presents some possible

extensions to Ada to make it easier to make library calls that accept callback parameters, and to make it easier to use thread local storage in Ada. Test results show that the libraries perform well in both native environments, and non-native environments such as the Mono/C# virtual machine. The paper suggests that a relatively simple library based approach is possible, and should be considered when evaluating syntax based proposals for programming languages, in that the ease and use of syntax should be an improvement over library based approaches, to be considered worthwhile.

References

[1] International Standards Organization (2012), *ISO IEC 8652:2012. Programming Languages and their Environments – Programming Language Ada*., Geneva, Switzerland.

[2] S. Michell, B. Moore, L. M. Pinho (2013), *Tasklettes – a Fine Grained Parallelism for Ada on Multicores*, International Conference on Reliable Software Technologies - Ada-Europe 2013, LNCS 7896, Springer.

[3] L. M. Pinho, B. Moore, S. Michell (2014), *Parallelism in Ada: status and prospects*, International Conference on Reliable Software Technologies - Ada-Europe 2014, LNCS 8454, Springer.

[4] T. Taft, B. Moore, L. M. Pinho, S. Michell (2014), *Safe Parallel Programming in Ada with Language Extensions*, High-Integrity Language Technologies conference (HILT 2014).

[5] B. Moore, S. Michell and L. M. Pinho (2013), *Parallelism in Ada: General Model and Ravenscar*, 16th International Real-Time Ada Workshop, York, UK.

[6] Working Draft, *Technical Specification for C++ Extensions for Parallelism*, available at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3960.pdf

[7] OpenMP Architecture Review Board (2013), *OpenMP Application Program Interface*, Version 4.0.

[8] Intel Corporation, *Cilk Plus*, https://software.intel.com/en-us/intel-cilk-plus

[9] TBB, *Threading Building Blocks*, at https://www.threadingbuildingblocks.org/

[10] Paraffin, *Paraffin Parallelism Libraries*, at http://paraffin.sourceforge.net

[11] Microsoft Corporation, *TPL*, https://msdn.microsoft.com/en-us/library/dd460717(v=vs.110).aspx

[12] Oracle Corporation, *Java Streams*, http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html

[13] C11 standard, http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf

[14] AI12-0189-1 *Loop Body as anonymous procedure*, at http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai12s/ai12-0189-1.txt?rev=1.

[15] R. D. Blumofe and C. E. Leiserson (1999), *Scheduling multithreaded computations by work stealing*, J. ACM, 46:720-748.

[16] B. Moore (2010), *Parallelism Generics For Ada 2005 and Beyond*, ACM SigAda 2010.

Parallel Reduction Lists

A Data Structure and Algorithm for Non-commutative Parallel Reduction with Bounded Storage

Brad Moore

General Dynamics, Canada, brad.moore@gdcanada.com

Abstract

This paper describes a generic data structure and associated algorithm that can be used by a parallelism framework to provide non-commutative reduction. All parallel reductions must be associative since the ordering of combination of results will need to be different than the sequential case if parallelism is to be achieved. Most parallel reductions however, such as addition, are also commutative which simplifies the reduction because results can be combined in any order and still determine the same result. There are a certain class of reduction operations such as concatenation and matrix chain multiplication, where the operation is not commutative. This paper describes an approach that allows the reduction to occur in parallel as part of the initial parallel processing, while also limiting the storage needed to be based on the number of executors assigned to the parallelism, rather than the number of reductions. As a result, the storage needed is typically smaller, and can be bounded based on the number of cores. Since the storage needed is small, it can be implemented without memory allocation from the heap. The algorithm and data structure can accommodate load balancing approaches such as Work Stealing, and Work Seeking, and has been implemented in the generic Paraffin libraries for Ada.

1 Introduction

In the computing world today, various parallelism frameworks exist [1 – 5], and a common need addressed by these frameworks is to provide a means to combine results from various parallel computations into a final result. An example of such a computation would be to compute the sum of integers from 1 to N. To provide parallelism, a divide and conquer approach is applied where the overall task is broken into several smaller ones working on different parts of the computation. Each part of the computation can be assigned to different executors which can then execute the sub-task on a different core of a multicore processor. An executor is conceptually a worker thread, which can accept items of work from the parallelism framework. Each executor has its own local copy of a partial result, in this case, an integer sum which it computes for its assigned sub-task. At some point, the different executors need to combine their results to produce the final sum. The simplest approach is to have the executor apply the reduction operation, which in this case, is addition, to the final result as soon as the executor completes its assigned work.

This simple approach works well for reduction operations such as addition, which are commutative operations, because the ordering of the combination does not matter, and will deterministically produce the same results each time the computation is repeated, assuming arithmetic overflow does not occur.

All reduction operations must be associative, if parallelism is desired because in order to achieve parallelism, the sequential ordering of combination must be altered to allow a divide and conquer approach. Most parallel reduction operations one typically encounters are also commutative including, addition, multiplication, min, max, set union, set intersection (and and or). There are some possible parallel reductions however that are not commutative. Some examples include concatenation and appending items to a list, and chain multiplication of matrices. For such problems, the approach described above will not work, because to achieve a deterministic result, the partial results need to be combined preserving the sequential ordering that would have been the case for sequential processing.

```
-- Concatenation example, a sorted result is desired
for Letter in 'A' .. 'Z' loop
   Alphabet := Alphabet.Append(Letter);
end loop;
```

To satisfy the need for non-commutative reductions, one approach would be to assign an ordinal value to each executor that reflects the ordering of the sequential computation, and then sequentially combine those results after the parallel processing is complete using the same ordering.

e.g., assuming 4 executors, the partial results before reduction might be:

```
Executor1: Result1="ABCDEF"
Executor2: Result2="GHIJLK"
Executor3: Result3="MNOPQRS"
Executor4: Result4="TUVWXYZ"

Final Result := Result1 || Result2 || Result3 || Result4
```

If the work is simply divided evenly between the cores, then a bounded array can be used to store the results, but the storage needs becomes more complex if more sophisticated parallelism algorithms are applied, such as load balancing strategies including Work Stealing [6], or

Work Seeking [7]. In such strategies, a given executor may get reused to offload work from other executors, once they become idle after processing their work items. It becomes apparent that in order to maintain the deterministic result, the storage needed to store the partial results is based on the number of reductions, not the number of executors, as each partial result will need to be combined separately using the sequential ordering.

One approach involves replacing the partial result array from above, with a tree like data structure. Further, since the number of reductions for a large number of loop iterations might be relatively large, a dynamic, heap-based tree structure might be needed to store these intermediate partial results.

A second observation is that the reduction operation likely occurs sequentially after the parallel processing. The reduction of the tree might itself be performed as a parallel operation, but that is an additional parallel operation which implies addition parallelism overhead. Typically the amount of processing needed for reduction is much less than that needed for the initial parallel processing, and the amount of processing to perform is not enough to overcome the overhead of a second parallelism operation, so in many cases, this reduction phase is best performed sequentially.

This paper describes an alternate approach where the reduction operation is performed in parallel as part of the initial parallel processing, without introducing the extra parallelism overhead associated with a second parallel operation. By the time the initial parallelism is complete, the reduction is also complete, which may offer some performance benefits over applying the reduction after the initial parallelism processing. The approach also allows the storage to be bounded based on the number of executors, rather than the number of reductions so that the storage requirements are typically much less than a tree based approach, where a stack based storage scheme can more readily be employed instead of a heap based one.

2 Reduction List – Data structure

The reduction list data structure is implemented as a bounded array of elements where each element is a node of a doubly linked list. It is a generic data structure where the element of each node in the list contains a partial result value of the reduction type that is associated with the parallel computation.

The number of elements in the list is equal to the number of executors + 1.

The number of executors typically equals the number of cores or is at least based on the number of cores typically, but it can be any number. The extra node in the list is used to hold the final reduction result. The data structure also maintains a free list which represents the executors that are currently idle looking for work. The data structure can be visualized from left to right where the leftmost node is allocated to the final result, and the nodes to the right of that node represent the executor whose ordinal number matches the index of the array. The reduction values of each element in the array is initialized to the identity value for the reduction operation. The identity value of a reduction operation is a value that when applied to an existing value does not alter the result. For addition, the identity value would be 0, since X + 0 = X. For multiplication, the identity value is 1, since X * 1 = X. For concatenation for a list, the identity value is an empty list, since an empty list concatenated to another list does not modify that other list.

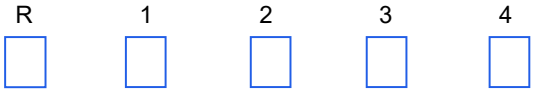
Associated also with the list, is a reduction operation. This is an operation of the form;

```
X := Reduce(L, R);
```

where the reduction result X is computed based on two inputs to be combined. If the result of interest to be combined is to be supplied as the left argument to the Reduce call, it is considered to be a “left” reduction, whereas if the result of interest is to be supplied as the right argument, it is considered to be a “right” reduction. For commutative reduction operations this distinction is not important, but for non-commutative reduction operations care must be taken to ensure that the results to be combined are passed in the proper order, in order to achieve the correct, deterministic results.

The data structure algorithm maintains two separate lists. One list is the doubly linked set of reduction values that have been computed, and possibly only partially reduced, and the other is a singly linked list of executors that are currently idle.

For example, for 4 executors, for a quad-core processor, the Reduction list data structure for the alphabet problem above might initially look like:



Idle list: 1 -> 2 -> 3 -> 4
Reduction list : R
Executor1 State: Idle
Executor2 State: Idle
Executor3 State: Idle
Executor4 State: Idle.

Where slot “R” represents the slot for the final result, and slots 1 to 4 represent the result for executors 1 to 4. All executors are initially idle and thus can be found in the free list.

3 Reduction List - Algorithm

The algorithm can be broken down into two separate operations.

1) Acquiring Work

2) Performing Reduction

3.1 Acquiring Work Operation

Initially the full workload to be processed is assigned to some set of executors. It might be assigned to a single executor as might be the case for certain work-stealing

approaches, or it might initially be distributed evenly across the executors, as is the case with Paraffin's work seeking approach.

Once the initial work has been assigned, any subsequent assignment of work works as follows:

1) The idle worker is removed from the free list and is doubly linked into the reduction list after the element from which it expects to acquire work.
2) The work from the element whose work is being acquired is divided such that the first half of the work remains in that same element, and the latter half is given to the worker that was previously idle.

Performing Reduction Operation

Once an executor has completed its assigned work, the processing is as follows:

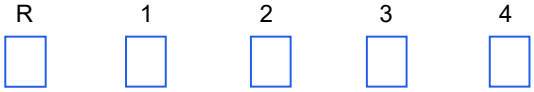
1) The idle worker does a left reduction into its own element of the list. The right value for the reduction is the existing value associated with that element.
2) The idle worker then performs a right reduction using the resulting value of the previous step into the element to the left in the reduction list.
3) The idle worker then unlinks its list element from the reduction list, and appends itself to the singly linked idle list.
4) The idle worker then proceeds to search for or request more work.

Determination of Done

Once the initial work has been assigned, the parallel operation is complete when all executors are in the idle list. At that point, the reduction list will only contain a single element, the first element in the data structure array, which holds the final reduction result.

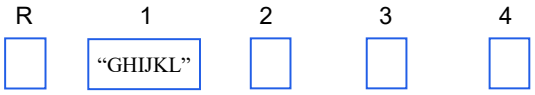
4 Illustrating by example

To illustrate how this works, reconsider the alphabet problem that was presented above. Initially it will be assumed that the workload is more or less evenly distributed amongst the executors. At this point, the state might look like the following:



Idle list: Empty
Reduction list: R <=> 1 <=> 2 <=> 3 <=> 4
Executor1 State: Work “ABCDEF”, Processing “ABCDEF”
Executor2 State: Work “GHIJKL”, Processing “GHIJKL”
Executor3 State: Work “MNOPQRS”,Processing “MNOPQRS”
Executor4 State: Work “TUVWXYZ”,Processing “TUVWXYZ”

Now let’s assume that executor 2 completes all of its assigned work while the other executors have only managed to process their first letter. Executor 2 first left reduces its result into slot 2, then right reduces that result into its left node which is slot 1. It then unlinks itself from the reduction list and appends to the idle list. At this point, the state would be:

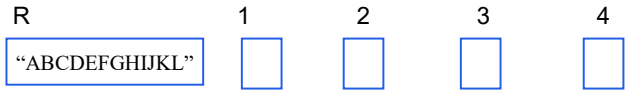


Idle list : 2
Reduction list : R <=> 1 <=> 3 <=> 4
Executor1 State: Work “ABCDEF”, Processing “BCDEF”
Executor2 State: Work “”, Processing “”
Executor3 State: Work “MNOPQRS”, Processing “NOPQRS”
Executor4 State: Work “TUVWXYZ”, Processing “UVWXYZ”

Now let’s assume that worker 1 completes its work, while workers 3 and 4 only manage to process a single letter. At this point executor1 does a left reduction into its slot, and then does a right reduction into the slot that is linked to the left of itself, which in this case is the final result element.

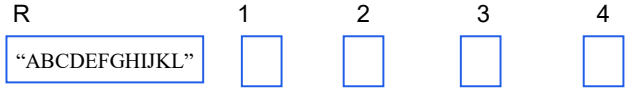
It then unlinks itself from the reduction list and appends itself to the idle list.

At this point, the state would be:



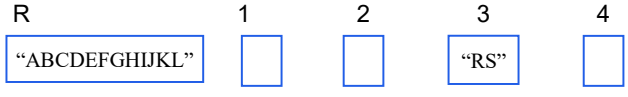
Idle list : 2 -> 1
Reduction list : R <=> 3 <=> 4
Executor1 State: Work: “”, Processing “”
Executor2 State: Work: “”, Processing “”
Executor3 State: Work: “MNOPQRS”, Processing “OPQRS”
Executor4 State: Work: “TUVWXYZ”, Processing “VWXYZ”

Now let’s assume that worker 2 has determined that it can acquire work from worker 3. First it removes itself from the idle list, and then inserts itself into the doubly linked reduction list after the executor 3. It then splits the remaining work for executor 3 with itself, taking the second half of the work. At this point, the state would be:



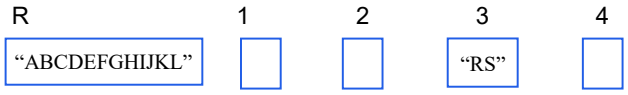
Idle list: 1
Reduction list: R <=> 3 <=> 2 <=> 4
Executor1 State: Work: “”, Processing “”
Executor2 State: Work: “RS”, Processing “RS”
Executor3 State: Work: “MNOPQ”, Processing “OPQ”
Executor4 State: Work: “TUVWXYZ”, Processing “VWXYZ”

Now let’s assume worker 2 completes its work. It left reduces into slot 2, then right reduces to the left element which is executor 3 in slot 3, and then removes itself from the reduction list and appends to the idle list. Now we have:



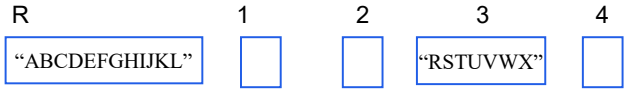
Idle list : 1 -> 2
Reduction list : R <=> 3 <=> 4
Executor1 State: Work: “”, Processing “”
Executor2 State: Work: “”, Processing “”
Executor3 State: Work: “MNOPQ”, Processing “OPQ”
Executor4 State: Work: “TUVWXYZ”, Processing “VWXYZ”

Now let’s assume executor 1 acquires work from executor 3. That leaves:



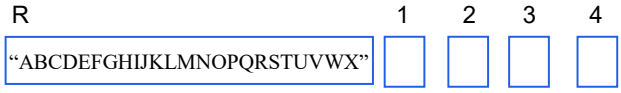
Idle list : 2
Reduction list : R <=> 3 <=> 4 <=> 1
Executor1 State: Work: “YZ”, Processing “YZ”
Executor2 State: Work: “”, Processing “”
Executor3 State: Work: “MNOPQ”, Processing “OPQ”
Executor4 State: Work: “TUVWX”, Processing “VWX”

Now let’s assume worker 4 finishes. Using the same processing described above, this leaves;



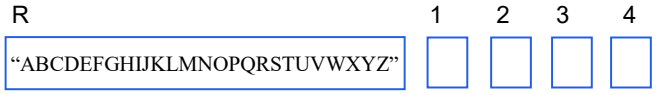
Idle list : 2 -> 4
Reduction list : R <=> 3 <=> 1
Executor1 State: Work: “YZ”, Processing “YZ”
Executor2 State: Work: “”, Processing “”
Executor3 State: Work: “MNOPQ”, Processing “OPQ”
Executor4 State: Work: “”, Processing “”

Now executor 3 finishes while executor 1 processes a letter leaving:



Idle list : 2 -> 4 -> 3
Reduction list : R <=> 1
Executor1 State: Work: “YZ”, Processing “Z”
Executor2 State: Work: “”, Processing “”
Executor3 State: Work: “”, Processing “”
Executor4 State: Work: “”, Processing “”

Now executor 1 completes which leaves the expected final result:



Idle list : 2 -> 4 -> 3 -> 1
Reduction list : R
Executor1 State: Work: “”, Processing “”
Executor2 State: Work: “”, Processing “”
Executor3 State: Work: “”, Processing “”
Executor4 State: Work: “”, Processing “”

At this point, all the executors are idle, the parallel computation is complete, and the reduction is also complete and can be extracted from the reduction list data structure.

5 Measurements

The Paraffin libraries include a series of tests for various problems. To illustrate the performance overhead associated with the use of the Reduction List, the example of calculating the sum of 64 bit integers/ 128 bit floats from 1 to 400_000_000 was used. All the results that begin with “Ordered” use the reduction list data structure, even though its use is unnecessary for addition reductions, since addition is a commutative operation. The other tests involve comparable versions that do not use reduction lists. Paraffin also has bindings to OpenMP (and Cilk) which uses their respective runtimes to provide the executors, but where Paraffin manages the reductions. Some results from the Paraffin OpenMP bindings are included here, for additional comparison. In these versions of the bindings, there is no usage of the Reduction List data structures. From these results, for this particular test, we see that there does not appear to be any significant overhead with using the reduction list, and in fact, in this particular run, some of the best times come from the library calls that do use the reduction lists, although there is variations in times over subsequent runs, and the best performer for a given test is not always the same library call. This is only a representative run. Averaging over many runs might allow one to draw further conclusions, but from numerous executions, it can be seen that these results are typical results.

Two test runs were collected, the first from a Ubuntu desktop environment running on an AMD Athlon II X4 635 Processor, the second run is for a Raspbery pi 2, running Raspbian.

```
********** Parallel Framework Test **********
  Physical Processors= 4
  Platform  = Ubuntu Desktop
  Executors = 4
  OS        = LINUX
  Processor = AMD Athlon(tm) II X4 635 Processor

Integer Reduction of Sum from 1 to 400_000_000

Sequential:                 Elapsed = 00:00:01.01
Work Sharing:               Elapsed = 00:00:00.26
Ordered Work Sharing:       Elapsed = 00:00:00.25
OpenMP Static:              Elapsed = 00:00:00.27
OpenMP Dynamic:             Elapsed = 00:00:00.25
OpenMP Guided:              Elapsed = 00:00:00.24
Work Seeking:               Elapsed = 00:00:00.23
Ordered Work Seeking:       Elapsed = 00:00:00.24
Work Stealing:              Elapsed = 00:00:00.25
Ordered Work Stealing:      Elapsed = 00:00:00.25
Pooled Work Sharing:        Elapsed = 00:00:00.27
Ordered Pooled Work Sharing: Elapsed = 00:00:00.25
Unbounded Pooled Work Sharing:
                            Elapsed = 00:00:00.26
Ordered Unbounded Pooled Work Sharing:
                            Elapsed = 00:00:00.25
Pooled Work Seeking:        Elapsed = 00:00:00.24
Ordered Pooled Work Seeking: Elapsed = 00:00:00.24
Pooled Work Stealing:       Elapsed = 00:00:00.27
Ordered Pooled Work Stealing:Elapsed = 00:00:00.26
```

```
*** Floating Point Addition Reduction Tests ***

Sequential Addition:         Elapsed = 00:00:03.72
Work Sharing Addition:       Elapsed = 00:00:00.97
Ordered Work Sharing:        Elapsed = 00:00:00.99
OpenMP Static:               Elapsed = 00:00:01.00
OpenMP Dynamic:              Elapsed = 00:00:00.98
OpenMP Guided:               Elapsed = 00:00:00.99
Work Seeking:                Elapsed = 00:00:00.98
Ordered Work Seeking:        Elapsed = 00:00:00.96
Work Stealing:               Elapsed = 00:00:00.99
Ordered Work Stealing:       Elapsed = 00:00:00.98
Pooled Work Sharing:         Elapsed = 00:00:00.98
Ordered Pooled Work Sharing: Elapsed = 00:00:00.98
Pooled Work Seeking:         Elapsed = 00:00:00.96
Ordered Pooled Work Seeking: Elapsed = 00:00:00.94
Pooled Work Stealing:        Elapsed = 00:00:00.95
Ordered Pooled Work Stealing:Elapsed = 00:00:00.98

********** Parallel Framework Test **********
  Physical Processors= 4
  Executors = 4
  Platform  = Raspbery pi 2
  OS        = LINUX (Raspbian)
  Processor = ARMv7 Processor rev 5 (v7l)

Integer Reduction of Sum from 1 to 400_000_000

Sequential:                  Elapsed = 00:00:18.76
Work Sharing:                Elapsed = 00:00:01.96
Ordered Work Sharing:        Elapsed = 00:00:01.99
Work Seeking:                Elapsed = 00:00:01.97
Ordered Work Seeking:        Elapsed = 00:00:01.98
Work Stealing:               Elapsed = 00:00:02.01
Ordered Work Stealing:       Elapsed = 00:00:01.97
Pooled Work Sharing:         Elapsed = 00:00:01.98
Ordered Pooled Work Sharing: Elapsed = 00:00:01.99
Unbounded Pooled Work Sharing:
                             Elapsed = 00:00:02.02
Ordered Unbounded Pooled Work Sharing:
                             Elapsed = 00:00:01.96
Pooled Work Seeking:         Elapsed = 00:00:01.98
Ordered Pooled Work Seeking: Elapsed = 00:00:01.98
Pooled Work Stealing:        Elapsed = 00:00:02.00
Ordered Pooled Work Stealing:Elapsed = 00:00:01.98

*** Floating Point Addition Reduction Tests ***

Sequential Addition:         Elapsed = 00:00:37.94
Work Sharing Addition:       Elapsed = 00:00:06.42
Ordered Work Sharing:        Elapsed = 00:00:06.40
Work Seeking:                Elapsed = 00:00:06.41
Ordered Work Seeking:        Elapsed = 00:00:06.39
Work Stealing:               Elapsed = 00:00:06.49
Ordered Work Stealing:       Elapsed = 00:00:06.41
Pooled Work Sharing:         Elapsed = 00:00:06.49
Ordered Pooled Work Sharing: Elapsed = 00:00:06.44
Pooled Work Seeking:         Elapsed = 00:00:06.38
Ordered Pooled Work Seeking: Elapsed = 00:00:06.41
Pooled Work Stealing:        Elapsed = 00:00:06.52
Ordered Pooled Work Stealing:Elapsed = 00:00:06.39
```

6 The Reduction List Ada package specification

The following shows the structure for the data structure and intended usage as realized in Paraffin [2].

```
generic
  type Element_Type is private;
  -- The type of the reduction result

  with function Reducer
       (Left, Right : Element_Type)
       return Element_Type;
  Identity_Value : Element_Type;

package Parallel.Functional_Reducing_Linked_List is

  type List (Worker_Count : Positive_Worker_Count)
    is limited private;

  subtype Donor_Id is Worker_Count_Type;
  subtype Effective_Worker_Id is Donor_Id
    range 1 .. Donor_Id'Last;

  type Cursor is private;

  function Create
    (Worker_Count : Positive_Worker_Count)
    return List;
   -- Returns a list will all nodes pre-linked into
  -- the list. This is useful for iterative
  -- generics which start with all work initially
  -- assigned to workers. For recursive generics,
  -- where work is assigned to one worker and
  -- grows recursively it is better to start with
  -- an empty list, which is what you start with
  -- if you dont make this call.

  function To_Cursor
    (Worker : Donor_Id) return Cursor;

  procedure Reduce
    (Container : in out List;
     Item : Element_Type;
     Position : Cursor);
  -- Performs a reduction of the value of 'Item'
  -- into the workers node,
  -- as the left operand using the current value
  -- of the node as the right operand. If there is
  -- a node to the left of the workers node then
  -- the resulting value is then reduced into the
  -- node to the immediate left of the workers
  -- node as the left operand where the right
  -- operand is the current value of that node.
  -- The workers node is then unlinked from the
  -- list.

  procedure Insert_Right
    (Container : in out List;
     Item, Position : Cursor);
  -- Inserts a workers node (the 'Item') right of
  -- the specified position in the reduction list.
  -- This is a non-blocking call.

  function Value
    (Container : List;
```

```
      Position : Cursor) return Element_Type;
   -- Returns the value of an element at a cursor

   procedure Result
     (Container : in out List;
      Reduction_Result : out Element_Type);
   -- Returns the final result of the reduction
   -- once all workers have reported their result.
   -- The call blocks until all the work is done.

   procedure Worker_Failed
     (Container : in out List);

private

   type Node_Type;
   type Node_Access is access all Node_Type;

   type Element_State is (Deleted, Available);

   type Node_Type is
     record
        Element : Element_Type;
        Next    : Node_Access;
        Prev    : Node_Access;
        State   : Element_State;
     end record;

   type Element_Array is
     array (Worker_Count_Type range <>) of
      aliased Node_Type;

   protected type List
     (Worker_Count : Positive_Worker_Count) is

      procedure Reduce
        (Item : Element_Type;
         Source : Positive_Worker_Count);

      procedure Insert_Right
        (Item, Position : Positive_Worker_Count);

      procedure Setup_All_Reductions;
      -- Initializes the list with all workers
      -- having work assigned.
      -- This is useful for initializing the list
      -- for the iterative generics, where all
      -- workers are loaded with work up front.

      entry Result (Item : out Element_Type);

      function Value (Index  : Worker_Count_Type)
        return Element_Type;

      procedure Worker_Failed;

   private

      Outstanding_Reductions : Worker_Count_Type
        := 0;
      Initialized : Boolean := False;
      Elements : Element_Array (0 .. Worker_Count)
       := (0 =>
            (Element => Identity_Value,
             Next => null,
             Prev => null,
             State => Available),
          others =>
            (Element => Identity_Value,
             Next => null,
             Prev => null,
             State => Deleted));
   end List;

   type Cursor is new Donor_Id;

   pragma Inline (To_Cursor);
end Parallel.Functional_Reducing_Linked_List;
```

Conclusion

This paper describes a data structure and algorithm that allows a parallelism framework to compute reductions, including non-commutative reductions in parallel during the main parallel processing, while also requiring minimal storage that may be suitable for placement on the stack, without requiring any heap allocation. It has been used successfully in Paraffin and has been shown to not introduce significant overhead, compared to other libraries of Paraffin that do not use the data structure, for reduction operations that are commutative.

References

[1] TBB, *Threading Building Blocks*, at https://www.threadingbuildingblocks.org/

[2] Paraffin, *Paraffin Parallelism Libraries*, at http://paraffin.sourceforge.net

[3] Working Draft, *Technical Specification for C++ Extensions for Parallelism*, available at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3960.pdf

[4] Intel Corporation, *Cilk Plus*, https://software.intel.com/en-us/intel-cilk-plus

[5] OpenMP Architecture Review Board (2013), *OpenMP Application Program Interface*, Version 4.0.

[6] B. Moore (2010), *Parallelism Generics For Ada 2005 and Beyond*, ACM SigAda 2010.

[7] R.D. Blumofe and C.E. Leiserson (1999), *Scheduling multithreaded computations by work stealing*, J. ACM, 46:720-748.

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: **www.ptc.com/developer-tools**

Summary of the 18th International Real-Time Ada Workshop

***Stephen Michell* [*], *Jorge Real* [§]**

[*] *Maurya Software, Canada*

[§] *Universitat Politècnica de València, Spain*

Abstract

The 18th International Real-Time Ada Workshop was held in Benicàssim, near Valencia, Spain. The main focus was on developing proposals that relate to real-time and high-integrity systems support in Ada. The workshop was very successful both in refining existing proposals and in identifying important new ones. The delegates also had a thoroughly enjoyable time and are very grateful to the organizers, for all their efforts.

1 Introduction

The 2016 edition of the International Real-Time Ada Workshop series was held in Benicàssim, Spain April 11-13 2016. The workshop discussed topics of concern to the real-time Ada community, in a relaxing and working environment.

Fig 1 - Benicàssim

The workshop tackled six main topic areas:

1. Deadline floor protocol issues;

2. Parallel programming issues;

3. Language issues;

4. Experience;

5. Language profiles; and

6. Time-based language vulnerabilities.

The workshop was attended by participants from Canada, Italy, Norway, Portugal, Spain, United Kingdom and United States of America.

Fig 2 – Guess who's who?

This paper provides a brief summary of the discussions and decisions taken at the workshop. For further details on the topics and discussions, the reader is referred to the workshop proceedings, to be published in the August 2016 issue of the Ada Letters.[1]

2 Deadline Floor

This session examined issues associated with the deadline floor protocol as first proposed at IRTAW 2013, and issues identified at that workshop. The position paper considered was Burns and Wellings, "The deadline floor protocol and Ada" [1].

The workshop noted that the proposal would imply changes to the current support for EDF scheduling in Ada, such as the addition of explicit types and objects to represent deadlines. The workshop preferred extending the current package Ada.Dispatching.EDF.

The workshop also recommended the deprecation of the Stack Resource Protocol and recommended support of a deadline floor aspect for protected objects. It was believed that the current task aspect Relative_Deadline could be overloaded to support also protected objects.

3 Parallel Programming

The Multicore/Parallel Processing session examined issues associated with the addition of syntax to Ada to effectively

[1] Editor note: as usual the Ada User Journal will provide the detailed session summaries of the workshop in a forthcoming issue.

manage parallel computation on multicore processors. The papers considered were: Michell, Pinho, Moore and Taft, "Constraints on the Use of Executors in Real-time Systems" [2], and Taft, Moore, Pinho and Michell, "Reduction of parallel computation in the parallel model for Ada" [3].

The majority of the effort was in solidifying constraints on parallel processing in real-time systems. All systems are concerned with the relative efficiency and correctness of parallel computation vs. strictly sequential computation, but real-time systems add dimensions of timing determinism, overruns, and scheduling. The workshop determined that

- Priority changes should not be permitted within parallelizable code. Any such operation should be deferred;
- Timing events should not be called or serviced within parallelizable code;
- Issues around the use of Set_CPU for tasks within parallelizable code are complicated enough that a new Set_CPU subprogram should be defined that permits the allocation of a task to a single CPU and its tasklets to a set of CPU's;
- Work stealing or parent stealing leads to code that cannot be statically analysed using current methods and tools;
- Programmers should be able to specify the maximum number of allowed executors and of active executors for the execution of tasklets, but there should be no mechanism to name or control executors;
- Tasklet control, in the sense of allocating "chunks" to tasklets during parallel execution, is essential;
- At this time, lack of analysis of highly parallel systems means that they should not be used in hard real-time partitions.

Discussions were held on the reduction proposals from the second paper [3]. There was general support for the notion of syntax to specify and control parallelism in Ada (as opposed to strictly library-based solutions). There was also support for the programmer to control aspects in the "map" and "reduce" aspects of parallelization.

4 Language Issues

The goal of this session was to discuss and, if appropriate, generate Ada Interpretations for several language related issues presented to the workshop [4]:

- Extension of Synchronous Task Control in order to allow the use of Suspension Objects by concurrent tasks.
- Inclusion of Synchronous Barriers in the Ravenscar profile.
- Addition of execution time timers and group budget support for interrupt handlers.
- Issues on High-Integrity Dynamic Memory Management.

The issue of synchronous task control was based on the expectation that multiple tasks co-ordinate access to a suspension object. The workshop confirmed that only a single task can suspend on a suspension object, and that it should be a bounded error for more than one task to suspend on a single object.

The workshop considered the inclusion of synchronous barriers in the definition of Ravenscar. While there was a general consensus that synchronous barriers may be useful in Ravenscar systems, it was decided that more feedback is needed from industrial use of Ravenscar on multiprocessor systems before they could be included.

The workshop discussed a proposal by Kristoffer Nyborg Gregertsen, "Revising the Ada timers and group budgets to support execution time control for interrupt handling" [5]. The paper was concerned with potential overruns in a real-time system due to interrupts and system level events. The workshop agreed that library mechanisms could be employed to cure this issue, but syntax-based solutions are superior. There was insufficient support for the proposals to recommend changes to the Ada language at this time.

During the discussions of dynamic memory allocation, triggered by the paper by Wellings, Cholpanov and Burns, "Implementing Safety-Critical Java Missions in Ada" [6], significant issues were identified related to allocation, management and deallocation of the memory allocated to objects, such as accessed objects and task stacks. There should be an AI raised on this topic.

5 Experience

The experience session considered the proposal from Real, Sáez and Crespo, "Combined Scheduling of Time-Triggered Plans and Priority Scheduled Task Sets" [7]. This paper proposed mechanisms to combine priority-based and time-triggered scheduling. The approach was well received. It grants minimum delays for selected tasks (scheduled by a highest-priority time-triggered scheduler, driven by timing events), while it also supports the execution of priority-based tasks with less strict jitter requirements. Several programming patterns were also proposed, ranging from a simple time-triggered task to more complex subtask decompositions typical of control systems. Discussion led to the suggestion of additional patterns, such as breaking a long-running TT task into segments, and other subtask decompositions.

The workshop provided suggestions for further research, such as exploring a Ravenscar implementation that would ease certification; or considering the integration of this approach in a more general real-time utilities framework, such as the ones considered in previous editions of the workshop.

6 Language Profiles

The "Profiles" session examined various Ada profiles, both official and unofficial, that are being used in current

practice. The goal was to determine the desirability of formalizing language profiles, as was done with great success for the Ravenscar Tasking Profile. The position paper from Garrido, Lacruz, Zamorano and de La Puente, "In Support of Extending the Ravenscar Profile" [8], supported extensions to Ravenscar. The workshop considered extensions to Ravenscar, as well as other profiles that could be created for specialized programming in Ada.

There was strong resistance to extending Ravenscar in ways that would reduce determinism, such as permitting multiple entries in a protected object or permitting multiple tasks to queue on an entry queue. It was recognised that there is a real demand to extend Ravenscar in order to simplify the programming effort, and that there will be vendor-specific profiles provided that essentially do this extension.

It was strongly felt that, if a profile is developed that includes "Ravenscar" in its name, the profile should not remove any functionality of the "base" profile, and should be essentially the same as the base profile. For example, the addition of Earliest Deadline First scheduling to Ravenscar should not qualify as an extension. Such a rule should apply to any profiles developed and later extended.

7 Time Vulnerabilities

The "Time Vulnerabilities" session focussed on a paper submitted by Stephen Michell, "Time Issues in Programs Vulnerabilities for Programming Languages and Systems" [9].

ISO's WG 23 is amending TR 24772 "Guidance on avoiding programming language vulnerabilities". As part of that process, WG 23 is identifying vulnerabilities that have not been previously captured, either by WG 23, or by other organizations such as CWE, CERT, or MISRA. While WG 23 identified some concurrency vulnerabilities in edition 2 of TR 24772, WG 23 had not considered any issues related to the management or use of clocks and time in programs.

The document that initiated this session identified issues associated with the use of different clocks or time bases in a system; representation of time (including granularity and fixed word sizes); perceptions of the passage of time; missed deadlines due to timing errors; and time effects due to virtualization.

The workshop identified three high-level vulnerabilities to submit for the considerations of WG 23. At its teleconference after the workshop, WG 23 decided to incorporate these vulnerabilities, but to make them all application vulnerabilities and not language-based vulnerabilities.

References

[1] Burns, A., Wellings, A. (2016), *The deadline floor protocol and Ada*, Proceedings of the 18th International Real-Time Ada Workshop, Ada Letters, August 2016, ACM New York.

[2] Michell S., Pinho L.M., Moore B., Taft S.T. (2016), *Constraints on the Use of Executors in Real-time Systems*, Proceedings of the 18th International Real-Time Ada Workshop, Ada Letters, August 2016, ACM New York.

[3] Taft S.T., Moore B., Pinho L.M., Michell S. (2016), *Reduction of Parallel Computation in the Parallel Model for Ada*, Proceedings of the 18th International Real-Time Ada Workshop, Ada Letters, August 2016, ACM New York.

[4] Burns, A., Wellings, A. (2016), *Synchronous Task Control and Synchronous Barriers*, Proceedings of the 18th International Real-Time Ada Workshop, Ada Letters, August 2016, ACM New York.

[5] Gregertsen, K.N. (2016), *Revising the Ada timers and group budgets to support execution time control for interrupt handling*, Proceedings of the 18th International Real-Time Ada Workshop, Ada Letters, August 2016, ACM New York.

[6] Wellings, A., Cholpanov, V., Burns, A. (2016), *Implementing Safety-Critical Java Missions in Ada*, Proceedings of the 18th International Real-Time Ada Workshop, Ada Letters, August 2016, ACM New York.

[7] Real, J., Sáez, S., Crespo, A. (2016), *Combined Scheduling of Time-Triggered Plans & Priority Scheduled Task Sets*, Proceedings of the 18th International Real-Time Ada Workshop, Ada Letters, August 2016, ACM New York.

[8] Garrido J., Lacruz B., Zamorano J., de La Puente J.A. (2016), *In Support of Extending the Ravenscar Profile*, Proceedings of the 18th International Real-Time Ada Workshop, Ada Letters, August 2016, ACM New York.

[9] Michell, S., *Time Issues in Programs Vulnerabilities for Programming Languages and Systems*, Proceedings of the 18th International Real-Time Ada Workshop, Ada Letters, August 2016, ACM New York 2016.

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 in Sweden

attn. Rei Stråhle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.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