ADA USER JOURNAL

Volume 39
Number 2
June 2018

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.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 issue of the Ada User Journal marks an important milestone in the life of the Journal and the relation with its sibling publication in the USA, Ada Letters, as both simultaneously publish the proceedings of the International Real-Time Ada Workshop (IRTAW 2018), which took place last April. This is the result of a general agreement made between Ada-Europe and ACM SIGAda to strengthen the ties between both publications, both allowing joint publication of contents, as well as facilitating to Ada Letters the material that the AUJ usually publishes. This agreement was first put into place last December, when Ada Letters re-published three papers from the Industrial Track of Ada-Europe 2017, which had been published by the AUJ during 2017. Together with this joint publication, these are important steps which will strengthen both publications, and, consequently, the communities they serve.

As for the contents themselves. The IRTAW series of workshops is the primer forum for discussion on Ada's concurrency model and real-time capabilities, discussing not only the available Ada support in these topics, but mainly future evolutions of the language. It is thus a very rich discussion forum, where papers are not presented, but challenged and discussed. Therefore, the proceedings include not only the papers of the workshop, but also the summaries of the held discussions, around the topics of Parallel Programming, Multiprocessor Locking, Language Profiles, Time Triggered Scheduling, Deadline Floor Protocol, Ada 202X Language Issues and Clock support. The reader will find the summaries of these discussions in the first part of the proceedings.

The workshop contents then continue with the set of position papers. First, a paper from Luis Miguel Pinho, of the Polytechnic Institute of Porto, Portugal, and Eduardo Quiñones and Sara Royuela, from Barcelona Supercomputing Centre, Spain, proposing the integration of the parallel model of tasklets with OpenMP. Afterwards, Brad Moore, from General Dynamics Canada, discussing a new synchronous signals abstraction. The third paper, from Jorge Garrido, Juan Zamorano and Juan A. de la Puente, of Universidad Politécnica de Madrid, Spain, discusses protocols for accessing protected objects on multiprocessors platforms. This is followed by a proposal for a new Ada concurrency profile for small microcontrollers, from Mario Aldea-Rivas and Héctor Pérez-Tijero, of Universidad de Cantabria, Spain, and a study on Ravenscar and EDF, from Paolo Carletto and Tullio Vardanega, of the University of Padua, Italy. The next paper is from Jorge Real, Sergio Sáez, and Alfons Crespo, of the Universitat Politècnica de València, Spain, discussing issues on time-triggered scheduling under Ravenscar, and the workshop papers close with a paper from Kristoff Gregertsen, of SINTEF, Norway, on the clock support in Ada.

I definitely recommend the reading of the papers, and the summaries of discussion. Not only these provide important information on the current and potential future mechanisms to support concurrency (and parallelism), as well as real-time execution, in Ada, but also identify future topics of interest, to be analyzed in future workshops (next one provisionally planned for 2020).

As for the other contents of this issue of the AUJ, a special note to the announcement of the ACM SIGAda High Integrity Language Technology workshop, which will take place 5-6 November, in Boston, USA, co-located with the ACM SPLASH conference, and obviously, the announce of Ada-Europe 2019: the 24th International Conference on Reliable Software Technologies, which will take place 10-14 June 2019, in Warsaw, Poland, with the local organization of the Engineering Design Center (EDC), a partnership between General Electric (GE) Poland and the Polish Institute of Aviation, one of the largest engineering institutions in Europe. I hope to see you there!

And as usual, the reader will also encounter the information provided in the News Digest and Calendar sections, prepared by Jacob Sparre Andersen and Dirk Craeynest, their respective editors.

Luís Miguel Pinho
Porto
June 2018
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 Organisations

AdaCore French Connection

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Sat, 5 May 2018 23:44:11 +0200
Subject: Adacore French connection
Newsgroups: comp.lang.ada

Luke Guest asked:

> AdaCore was "spun out of NYU" and I still have no clue how it became a French company.

Around 1982, Philippe Kruchten was a professor at ENST (French engineering school), and had translated the (preliminary) ARM in French, so he had some connection with the people at NYU. He was invited to a sabbatical there.

I was also professor at ENST, so after 6 months he pulled me there too.

When we returned to France, the link was established, and we regularly sent professors for sabbatical and students as interns to NYU.

Some of them were the ones who founded the French branch of AdaCore when they returned home...

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]

Ada-Europe 2018 in Lisbon

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Sun, 1 Apr 2018 05:46:08 -0000
Subject: 23rd Int.Conf. Reliable Software Technologies, Ada-Europe 2018
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

Call for Participation

*** PROGRAM SUMMARY ***

23rd International Conference on Reliable Software Technologies
- Ada-Europe 2018

18-22 June 2018, Lisbon, Portugal

http://www.ada-europe.org/conference2018

Organized by Univ. Lisboa and Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN and the Ada Resource Association (ARA)

*** Online registration open ***

Early registration discount until May 14

*** Extensive info available on conference web site ***

*** Highly recommended to book your hotel ASAP ***

The 23rd International Conference on Reliable Software Technologies - Ada-Europe 2018 will take place in Lisbon, Portugal, from June 18 to 22. As per its traditional style, the conference will span a full week, including, from Tuesday to Thursday, three days of scientific, technical and industrial programs, along with tutorials and workshops on Monday and Friday.

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.

Extensive information is available on the conference web site, such as an overview of the program, the list of accepted papers and industrial presentations, and descriptions of workshops, tutorials, keynote presentations, and social events.

Also check the conference web site for registration, accommodation and travel information. The 16-page Advance Program brochure will shortly be available on the conference web site as well.

Quick overview

- Mon 18 & Fri 22: tutorials + workshops

- Tue 19 - Thu 21: core program

Proceedings

- published by Springer

- volume 10873 in Lecture Notes in Computer Science series

- will be available at conference

Program Chair

- António Casimiro, LASIGE/U. Lisboa, Portugal
casim at ciencias.ulisboa.pt

Keynote speakers

- Paulo Esteves-Veríssimo, University of Luxembourg, Luxembourg, "Security and Dependability Challenges of IT/OT Integration"

- Carl Brandon, Vermont Technical College, USA, "From Physicist to Rocket Scientist, and how to make a CubeSat that works"

- 3rd speaker to be confirmed

Workshops (full day)

- "Runtime Verification and Monitoring Technologies for Embedded Systems" Workshop (RUME 2018)

- 5th International Workshop on "Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering" (DeCPS 2018)

Tutorials (full day)

- "Recent Developments in SPARK 2014", Peter Chapin, Vermont Technical College, USA

- "Scheduling analysis of AADL architecture models", Frank Singhoff, Lab-STICC/UBO, France and Pierre Dissaux, Ellidiss Technologies, France

Tutorials (half day)

- "Access types and memory management in Ada 2012", Jean-Pierre Rosen, Adalog, France

- "Design and architecture guidelines for trustworthy systems", William Bail, The MITRE Corporation, USA

- "Numerics for the Non-Numerical Analyst", Jean-Pierre Rosen, Adalog, France
- "Requirements development for safety and security critical systems", William Bail, The MITRE Corporation, USA
- "Writing Contracts in Ada", Jacob Sparre Andersen, JSA Research & Innovation
- "Introduction to Libadalang", Raphaël Amiard and Pierre-Marie de Rodat, AdaCore, France
- "Unit-testing with Ahven", Jacob Sparre Andersen, JSA Research & Innovation
- "Frama-C, a Framework for Analysing C Code", Julien Signoles, CEA LIST, France

Papers and Presentations

- 10 refereed technical papers and 4 presentations in sessions on Safety and Security, Ada 202X, Handling Implicit Overhead, Real-time Scheduling, New Application Domains
- 12 industrial presentations and experience reports in sessions on Ada in Industry, Space Systems, V&V of Safety-Cricital Software, Software Methodologies
- submissions by authors from 19 countries, and accepted contributions from Austria, France, Germany, Italy, Norway Poland, Portugal, South Korea, Spain, Sweden, Switzerland, UK, USA

Vendor exhibition and networking area

- area features exhibitor booths, project posters, reserved vendor tables, and general networking options
- several companies already committed; others expected to confirm soon
- vendor presentation sessions in core program

Social events

- each day: coffee breaks in the exhibition space and sit-down lunches offer ample time for interaction and networking
- Tuesday evening: Ada-Europe General Assembly, followed by Welcome Reception; location will be announced in April
- Wednesday evening: transportation to restaurant "A Casa do Bacalhau", for the traditional Ada-Europe Conference Banquet; the name means "The House of the Codfish", so it is not too difficult to guess what is their speciality
- Best Paper and Best Presentation awards will be handed out

Registration

- online registration is open at <http://www.ada-europe.org/conference2018/registration.html>
- early registration discount up to Monday May 14, 2018
- additional discount for academia, Ada-Europe, ACM SIGAda, SIGBED and SIGPLAN members
- student discounts are available
- registration includes copy of printed proceedings at event
- includes coffee breaks and lunches
- three day conference registration includes all social events
- tutorial fees reduced when taken together with 3-day conference
- payment possible by credit card or bank transfer
- see registration page for all details

Promotion

- recommended Twitter hashtags: #AdaEurope and/or #AdaEurope2018
- 16-page Advance Program brochure soon available online
- support Ada-Europe 2018 with promotional poster available at <http://www.ada-europe.org/conference2018/posters/AE2018_poster.png>

Please make sure you book accommodation as soon as possible.

Lisbon will be very busy in that week.

For more info and latest updates see the conference web site at <http://www.ada-europe.org/conference2018>.

We look forward to seeing you in Lisbon in June 2018!

[See also “Ada-Europe 2018 in Lisbon”, AUJ 39-1, p. 8. —sparre]

Ada-related Resources

Ada on Social Media

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Fri May 4 2018
Subject: Ada on Social Media

Ada groups on various social media:

- LinkedIn:	2_706 members	[1]
- Reddit:	1_825 readera	[2]
- StackOverflow:	1_000 followers	[3]
- Google+:	764 members	[4]
- Freenode	88 participants	[5]
- Gitter:	55 people	[6]
- Twitter:	10 tweeters	[7]

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

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

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

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

[5] #Ada on irc.freenode.net

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

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

[See also “Ada on Social Media”, AUJ 39-1, p. 9. —sparre]

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Fri May 4 2018
Subject: Repositories of Open Source software

GitHub:	2_018 repositories	[1]
	503 developers	[2]
	2_013 issues	[3]
Rosetta Code:	645 examples	[4]
	33 developers	[5]
	0 issues	[6]

Sourceforge: 265 projects [7]

BlackDuck OpenHUB: 206 projects [8]

Bitbucket: 94 repositories [9]

Codelabs: 45 repositories [10]

AdaForge: 8 repositories [11]

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

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

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

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

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

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

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

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

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

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

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

[See also “Repositories of Open Source Software”, AUJ 39-1, p. 9. —sparre]

Ada and Software Engineering Library

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Tue, 8 May 2018 17:02:55 -0000
Subject: Re: How to Download From PAL ?
Newsgroups: comp.lang.ada

> [...]
http://archive.adaic.com/ase/index.html

[...]

A blast from the past... ;-)

What's on the AdaIC is a later version of the "Public Ada Library (PAL)" under it's (then) newer name of "Ada and Software Engineering Library Version 2 (ASE2)" from October 2000.

Randy fetched the copy for the AdaIC from the Ada-Belgium FTP server where we had put it up for everyone interested. IIRC, it is the final version of the "PAL" that was made.

I still have a box with most PAL versions on CD-ROM: Ada-Belgium did order many to be handed out to participants at our events at the time...

FWIW, the Ada-Belgium ftp server is long gone, but the original ASE directory is still available at: http://www.cs.kuleuven.be/~dirk/ada-belgium/archive/ase/

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 8 May 2018 15:22:02 -0500
Subject: Re: How to Download From PAL ?
Newsgroups: comp.lang.ada

> [...]

Everything that the ARA got from AJPO has been on-line in the AdaIC archives since 1998 (see http://archive.adaic.com). We didn't get *everything*; some stuff not on the AdaIC site seems to have not been saved.

In any case, if you truly find this important, you might consider becoming an ARA contributor to help with the expenses associated with it.

Ada-related Tools

Mathpaqs

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Tue, 13 Mar 2018 09:39:47 -0700
Subject: Mathpaqs release 13-Mar-2018
Newsgroups: comp.lang.ada

Mathpaqs is a collection of mathematical packages in the Ada programming language.

What's new:

- Added special functions Beta and Phi in the numerics part (Beta_function, Test_Beta, Phi_function)
- Added the Beta distribution in the random distribution part (Generic_Random_Functions)

More information here: http://mathpaqs.sf.net/

[See also “Mathpaqs”, AUJ 36-3, p. 121. —sparre]

From: Vincent Diemunsch <vincent.diemunsch@gmail.com>
Date: Wed, 14 Mar 2018 07:54:13 -0700
Subject: Re: Mathpaqs release 13-Mar-2018
Newsgroups: comp.lang.ada

[...]

Just a few questions :

- why not use the standard Generic Real Arrays ?
- Are the G_Matrices stored in Fortran convention, i.e. Column Major Order ?
- Are they compatible with Blas & Lapack ?
- Are the SparseB matrices compatible with Matlab matrices (i.e. CSC matrices)
- Are both format interoperable ?
- Are they interchangeable, i.e. is there a common class type to manipulate them?

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Fri, 16 Mar 2018 10:59:51 -0700
Subject: Re: Mathpaqs release 13-Mar-2018
Newsgroups: comp.lang.ada

> - why not use the standard Generic Real Arrays ?

Good point! I'll update the web site with a remark from g_matrices.ads: "NB: For Ada 2005+ and real numbers implemented as floating-point numbers, it is better to use Ada.Numerics.Generic_Real_Arrays instead.". An usage of G_Matrices can be for complex, rational, "bignum"-rationals, etc. ...

> - Are the G_Matrices stored in Fortran convention, i.e. Column Major Order ?

By default not, but you can add a pragma Convention.

> - Are they compatible with Blas & Lapack ?

Some stuff in the lin_alg area is, but frankly it is a bit rusty (in my head)...

> - Are the SparseB matrices compatible with Matlab matrices (i.e. CSC matrices)

SparseB is just a helper for Sparse package (I did it in Ada 83 around end 1990's, it would be a private child package now (tbd)...). There are two bodies for sparseb: one standalone, one using Blas (sparseb.blas.adb)

> - Are both format interoperable ?

> - Are they interchangeable, i.e. is there a common class type to manipulate them?

I've used for research purposes, interchangeably:

- plain matrices,
- band matrices (a matrix with zeroes outside diagonals and the cells just above and below) and
- sparse matrices.

I did it through generics.

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Tue, 27 Mar 2018 14:27:19 -0700
Subject: Ann: Mathpaqs release 27-Mar-2018
Newsgroups: comp.lang.ada

[...]

What's new:

- Discrete_Random_Simulation: after linear - O(n) - and dichotomic (binary) - O(log(n)) - searches, added the alias method which is O(1). Concretely, the simulation of a discrete, finite random variable of any number of states is as fast as simulating a flip-or-coin !...
- Added Test_Discrete_Random_Simulation with timing and error measurement.

AdaYaml

From: Felix Krause <contact@flyx.org>
Date: Sat, 17 Mar 2018 10:51:18 +0100
Subject: Re: YAML parser?
Newsgroups: comp.lang.ada

> Thank you, everyone. I'll try to get in touch with the maintainer of the YAML web site to see if they can update the pate (I do not know if there are constraints of maturity)

You can ask me right here if you have concerns :).

Concerning the maturity, AdaYaml is tested with the comprehensive test suite of the YAML project. The suite is still in development, but has far above 200 tests. AdaYaml passes all of the test cases relevant for a user, with failures in details where we are trying to change things for the better in upcoming YAML 1.3.

The API is mostly mature, except for a possible minor change in tag handling that may happen depending on or decision process on 1.3. You can assume that AdaYaml parses all „real“ YAML 1.2 documents (i.e. not depending on a few edge cases not found anywhere in the wild) and the biggest incompatibility with 1.2 is currently that verbatim tags are not allowed:

```
!<tag:yaml.org,2002:str> # fails with lexer error

!!str  #identical semantics, parses fine
```

The second „incompatibility“ is that '@' is no longer reserved, but parsed as „annotation“. This will not break any existing documents as the character was previously reserved. Annotations are currently little more than a proof-of-concept and should not concern you – if you just reject any annotation tokens, you have a normal 1.2 structure. All my experimental code is in yaml-annotation_processor.gpr so if you just import yaml.gpr, you won't import experimental code.

I am always a bit lazy on crafting new releases. As I know now that there is interest in using this library again, I will wrap up the current state into a new release, as there have been some bug fixes since the last one.

From: Felix Krause <contact@flyx.org>
Date: Thu, 22 Mar 2018 21:38:13 +0100
Subject: ANN: AdaYaml 0.3.0
Newsgroups: comp.lang.ada

I have just released AdaYaml 0.3.0. As always, documentation is found on the website [1] and the release is available as tag of the GitHub repository [2].

The reason for this release is mainly that development is currently stagnant, but I wanted the new DOM API to be available in a release since it is pretty stable. Shortly after development, the DOM API has been tested with fuzzing by Lionel Matias [3]. Findings of these tests have been addressed.

This release fixes some bugs in the previous release and there are no breaking changes.

[1]: https://ada.yaml.io

[2]: https://github.com/yaml/AdaYaml/tags

[3]: https://blog.adacore.com/running-american-fuzzy-lop-on-your-ada-code

[See also “AdaYaml”, AUJ 38-4, p. 176. —sparre]

Gnoga

From: Pascal Pignard <p.p14@orange.fr>
Date: Sat, 17 Mar 2018 11:05:29 +0100
Subject: Sprite support with events.
Newsgroups: gmane.comp.lang.ada.gnoga

I've pushed on dev_1.4 branch more API to sprites:

- add Angle_Limit API for Sprites (it sets sprite angle limit for rotation with special effect: null, bounce or loop).
- add new sprite effects for events:
  - Inside_Event_Effect is sent when sprite is inside frame or angles limits and resets when fired.
  - Outside_Event_Effect is sent when sprite is outside frame or angles limits and resets when fired.
- Add new API: Fire_On_Frame, Frame_Effect, Fire_On_Angle, Angle_Effect, On_Message.

It's a very first version committed to request comments, tested API are included in test/pixi_sprite_test.adb.

Feel free to send code review:

https://sourceforge.net/p/gnoga/code/ci/dev_1.4/tree/components/pixi/src/

Spite possibilities are wide, I'll add API step by step, feel free to point out some API you want to be available.

[See also “Gnoga”, AUJ 38-3, p. 118. —sparre]

Ada_GUI

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sun, 18 Mar 2018 14:33:24 +0100
Subject: Ada-Oriented GUI
Newsgroups: comp.lang.ada

I've written on here before that I think the traditional register-callbacks-and-call-a-procedure GUI interface is a hack only suitable for sequential languages, and attempted to describe my idea of how a GUI for a concurrent language like Ada should work. Some have expressed the opinion that such an interface is not possible.

I've now written a very minimal GUI interface embodying my concepts. It implements text boxes and buttons, as that is the minimum for an example that actually does something faintly interesting. For an example, I've used the Random_Int demo from Gnoga.

I've also made a quick and dirty implementation of the interface on top of Gnoga, and a version of Random_Int using the interface. Though the Gnoga version of Random_Int is very simple and fairly easy to understand, this version is easier to understand.

Those interested can find Ada_GUI at

https://github.com/jrcarter/Ada_GUI

Those who like programming by extension won't like it.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 21 Mar 2018 09:25:10 +0100
Subject: Re: Ada-Oriented GUI
Newsgroups: comp.lang.ada

> [...]

Why do you believe that event-loop architecture is better than callback one? In the latter there at least no need to have an explicit loop and more importantly it can be made type safe if callback are primitive operations of interfaces.

As Randy already mentioned, your design is quite close to Windows GDK, but very different from other architectures.

In any case I do not see how this responds to major challenges of GUI design, e.g. structured filtering of events (widgets swallow events and re-emit higher-level ones) or inversion when a button emits events, but it is the handler that must process them, so the case-statement or dispatching choice constrained by button interface must be in the handler code, not in the button or its descendant.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 20 Mar 2018 16:34:56 -0500
Subject: Re: Ada-Oriented GUI
Newsgroups: comp.lang.ada

> [...]

I can't speak to all such systems, but the above is definitely not true of Windows Win32. The GUI can be on any thread that it likes, the restriction is that one needs to use a single thread to create windows and receive messages. (Any thread can do other operations.) We took advantage of this in the design of Claw, where there is a separate Ada task created solely for the purpose of managing the GUI. On top of this we built a fairly conventional OOP design where the GUI task dispatches to appropriate subprograms. This allows Ada programs fairly free access to the GUI from any task. (It also has the downside of bringing in tasking issues into any Claw program, even if there aren't any explicit tasks.)

It would be easy for Jeff to build his GUI design on top of Win32, it would be organized somewhat like Claw internally but using a different interface to interact with it.

PHCpack

From: gerdien.de.kruyf@gmail.com
Date: Fri, 30 Mar 2018 14:25:57 -0700
Subject: Re: how to copy complete column (or row) of matrix to another?
Newsgroups: comp.lang.ada

> [...]

There are lots of interesting Ada maths packages in here:

https://github.com/janverschelde/PHCpack

WinRT Bindings

From: alby.gamper@gmail.com
Date: Fri, 13 Apr 2018 02:05:39 -0700
Subject: Ada-WinRT bindings - Alpha release
Newsgroups: comp.lang.ada

I am pleased to announce the initial alpha release of the WinRt bindings for Ada.

It is available on GitHub at the following URL

https://github.com/Alex-Gamper/Ada-WinRT

Please feel free to raise issues/question/recommendations for improvements either via GitHub or here on comp.lang.ada.

AdaBase

From: John Marino <dragonlace.cla@marino.st>
Date: Sun, 15 Apr 2018 09:08:24 -0700
Subject: Re: libgnadeodbc
Newsgroups: comp.lang.ada

(Just picking a post to reply to)

Well, nobody mentioned my database interface project, so I'll just post the links again:

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

https://github.com/jrmarino/AdaBase

It supports MySQL, PostgreSQL and SQLite well. In less complex use cases, it should be possible to have an application use any backend rather than be locked to a specific database (if abstraction capabilities are used).

[See also “AdaBase”, AUJ 37-2, p. 74. —sparre]

PolyORB

From: Thomas Quinot <quinot@adacore.com>
Date: Wed, 18 Apr 2018 10:43:38 -0700
Subject: PolyORB now lives on Github
Newsgroups: comp.lang.ada

I am pleased to announce that PolyORB, AdaCore's versatile distribution middleware, now lives on Github. While it remains a fully supported AdaCore product, its source repository now lives under our Github org at https://github.com/AdaCore/polyorb

AdaCore has always been committed to involving the user community in the development of PolyORB. Over the past 15 years, many contributions from industrial as well as hobbyist users have been integrated, and community releases were previously made available in conjunction with GNAT GPL.

Today we are pleased to further this community engagement and renew our commitment to an open development process by making the PolyORB repository (including full history) available on Github. This will allow users of GNAT GPL to benefit from the latest developments and contribute fixes and improvements.

We look forward to seeing your issues and pull requests on this repository!

Formal Methods Toolkit

From: Maciej Sobczak <maciej@msobczak.com>
Date: Sun, 22 Apr 2018 23:42:29 -0700
Subject: Formal Methods Toolkit
Newsgroups: comp.lang.ada

I'm pleased to announce the availability of new tool:

http://inspirel.com/fmt/

FMT is a set of extension packages that turn Mathematica into software engineering IDE allowing to design models, analyze, visualize, verify and generate formally-proven Ada [*] source code.

[*] for those who cannot resist the temptation ;-), C and C++ are targeted, too.

Please check the gallery of screenshots to see some samples of what is possible:

http://inspirel.com/fmt/gallery.html

FMT is open-source and free.

LEA

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Thu, 26 Apr 2018 01:50:05 -0700
Subject: LEA - Lightweight Editor for Ada - Binary release v 0.65
Newsgroups: comp.lang.ada

LEA is a Lightweight Editor for Ada.

What's new:

- Find & Replace box is now fully functional
- First binary distribution with an integrated mini-compiler (HAC)

Features:

- multi-document
- multiple undo's & redo's
- multi-line edit, rectangular selections
- color themes, easy to switch
- duplication of lines and selections
- syntax highlighting
- parenthesis matching
- bookmarks

Currently available on Windows. Gtk or other implementations are possible: the LEA_Common[.*] packages are pure Ada, as well as HAC.

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

[See also “LEA”, AUJ 38-4, p. 178. —sparre]

Cortex GNAT RTS

From: Simon Wright <simon@pushface.org>
Date: Sat, 28 Apr 2018 17:13:12 +0100
Subject: ANN: Cortex GNAT RTS 20180419
Newsgroups: comp.lang.ada

There are three parallel releases at Github[1], for

- GNAT GPL 2016/GCC 6
- GCC 7
- GNAT GPL 2017

(three, because of changes to the interface between the compiler and the RTS).

There's not much user-visible change from the last GCC 7 release[2], except that all but 2k of free store is available for the heap (the 2k is used for startup and interrupt stack).

[1] https://github.com/simonjwright/cortex-gnat-rts/releases

[2] https://github.com/simonjwright/cortex-gnat-rts/releases/tag/r20171016

[See also “Cortex GNAT RTS”, AUJ 38-3, p. 119. —sparre]

Simple Components

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 2 May 2018 18:51:02 +0200
Subject: Inter-process communication in Ada with Simple Components v 4.28
Newsgroups: comp.lang.ada

The latest version 4.28 of Simple Components introduces inter-process communication primitives. The implementation does not rely networking and is based on shared memory and OS primitives available. Both Linux and Windows are supported. No configuration is required, no source code generation is used either. The interface packages are OS-independent.

The following synchronization and communication primitives are provided:

- Manual set/reset event;
- Pulse event;
- Re-entrant mutex. A mutex that can be seized by the same task several times without blocking;
- Shared object that can be accessed from different processes;
- FIFO, first-in, first-out queue with the ends in different processes;
- Blackboard for publishing updates in a way that do not block the publisher;
- Inter-process stream with the end points in different processes;
- Shared memory pool which can be allocated and freed from different processes. The pool supports references which can be converted forth and back access type. The pool reference can be exchanged between processes and stored in the shared memory;
- Process call service, that provides remote procedure call facilities. Both synchronous and asynchronous remote calls are supported. Synchronous calls can return back results or update arguments. Exceptions propagated at the callee's side are reported back to the caller and re-raised at the call point.
- Manager of the process call services that allow processes involving in RPC exchange to come and go dynamically.

The configuration of primitives is based on introspection of the Ada type that describes the shared environment. The primitives are merely components of the shared environment object. The coherence of the shared environment is checked when the process joins other processes.

The intended audience is Ada developers of servers, e.g. Gnoga users. Bug reports and feature requests are welcome.

P.S. The Distributed Systems Annex E can be supported if there is demand, provided some help with configuring GNAT to respect the pragma Remote Call

Interface and use the custom System.RPC.

[See also “Simple Components”, AUJ 38-4, p. 176. —sparre]

Ada Drivers Library: STM32l4xx Nucleo Support?

From: John McCabe <john@mccabe.org.uk>
Date: Sat, 5 May 2018 03:01:45 -0700
Subject: stm32l4xx nucleo and Ada Drivers Library.
Newsgroups: comp.lang.ada

I'm looking at using a NUCLEO-L476RG board, with some attachments, for a prototyping exercise. As it's a work thing it'll probably be coded in C++ but, as these boards are so cheap, I thought I might try to investigate replicating (at least) some of the functionality in Ada at home. Looking at the Ada Drivers Library, there seems to be a few STM32F devices and associated DISCOVERY boards supported, but no specific mention of the L series or NUCLEO.

This is the first time I'll have used the STM32 series, so I wondered if anyone with more experience might be able to give me some brief guidance on getting going. For example, would copying one of the STM32F device folders and trying to tweak it for the device I'm using be a sane starting point? (Although, when I say starting point, I imagine I'd just try to get a noddy do nothing program up and running before trying anything clever!)

PDF_Out

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Sun, 6 May 2018 01:21:11 -0700
Subject: Ann: Ada PDF Writer v.004
Newsgroups: comp.lang.ada

PDF_Out is an Ada package for producing easily and automatically PDF files, from an Ada program, with text, vector graphics, images (JPEG). You can produce automatically reports, invoices, tickets, charts, maps etc. from your Ada program. The Ada PDF Writer is free and open-source.

What's new:

- A fix by G. Cannone was integrated. PDF documents produced by PDF_Out pass now an online PDF validation test.

More information here: http://apdf.sf.net/

[See also “PDF_Out”, AUJ 37-2, p. 74. —sparre]

AUnit

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 10 May 2018 13:39:00 -0700
Subject: Re: Ada aunit examples
Newsgroups: comp.lang.ada

> Does anyone here have a good example of a project that uses AUnit? [...]

SAL (http://www.stephe-leake.org/ada/sal.html) has AUnit tests. It also has an aunit extension package Checks that makes it easier to write AUnit tests.

Ada-related Products

RVS^Acad

From: Rapita Systems
Date: Tue, 10 Apr 2018 09:14:37 +0000
Subject: RapiTimes Newsletter April 2018

[...]

Rapita in academia and RVS^Acad

At Rapita, we've always had strong ties with the academic environment. Building on our roots as a spin-off company from the University of York, we remain actively involved in research activities including collaboration with universities worldwide.

To extend our range of software verification solutions to meet the needs (and budget) of the academic environment, we've recently launched RVS^Acad, an academic version of the Rapita Verification Suite that is offered at a greatly discounted rate compared to the commercial version.

[...]

[See also “Rapita Verification Suite”, AUJ 37-2, p. 77. —sparre]

VectorCAST

From: Vector Software
Date: Tue Apr 24 2018
Subject: Vector Software Announces New Release of the VectorCAST 2018 Test Automation Platform
URL: https://www.vectorcast.com/news/vector-software-press-releases/2018/vector-software-announces-new-release-vectorcast-2018-test

Providence, RI USA, 2018-APR-24 – Vector Software, now part of Vector Informatik, the leading manufacturer of software tools, embedded components, and leading provider of services for the development of electronic systems, announced today the release of VectorCAST 2018. This latest release of VectorCAST contains many new features as well as numerous enhancements to existing functionality. For example, the user benefits from increased user-friendliness, testing efficiency, and collaboration between team members.

The VectorCAST embedded software testing platform is a family of products that automates testing activities across the software development lifecycle. The new product release of VectorCAST 2018, using the new Probe Point functionality, provides a simple way to dynamically instrument a complete application with blocks of code (Probe Points). This enables white-box testing, injection of faults and debugging of hard to repeat race conditions. The user gains access to an intuitive graphical editor, which enables him to extend his existing tool chain to create and manage VectorCAST Probe Points.

The new Component Coverage functionality supports customers with limited target resources by allowing users to break their application into multiple logical components and instrument each component in isolation. This feature is integrated with the system test automation features of VectorCAST/QA which automatically runs all tests against each component and combines the coverage results into a single report.

VectorCAST 2018 provides full support for the coupling analysis and verification required by the DO-178B and C standards for avionics certification. Static analysis is performed to identify the couples that exist in the implementation, and source code instrumentation is performed to ensure that each of the identified couples is tested. Additionally, a simple to use graphical component editor allows user to easily create a component definition that matches the application’s architecture.

VectorCAST 2018 enables test collaboration across the whole enterprise, through the introduction of many new extensions. The user profits of newly developed functions for test artifact sharing and maintenance, real-time analytics, and improved test automation.

For additional details about the release of VectorCAST 2018, please visit: www.vectorcast.com/vectorcast-2018.

From: Vector Software
Date: Tue May 1 2018
Subject: Vector Software Launches New Product Editions of its Award-Winning VectorCAST Test Automation Platform
URL: https://www.vectorcast.com/news/vector-software-press-releases/2018/vector-software-launches-new-product-editions-its-award

Providence, RI USA, 2018-MAY-1 – With the release of VectorCAST 2018, two editions, Professional and Enterprise, are now available to meet the specific needs of smaller and larger teams and projects. Vector Software is now part of Vector Informatik, the leading manufacturer of software tools and embedded components for the development of electronic systems.

With the two new Editions, the automated test platform VectorCAST 2018 is able to better adapt to the individual conditions of each customer, who, from now on, may choose between “Professional” and “Enterprise” editions.

VectorCAST 2018 Professionals intended for single users and small workgroups working on projects that require automated black-box and white-box unit and integration testing, as well as code coverage analysis.

VectorCAST 2018 Enterprise includes all of the features of VectorCAST 2018 Professional plus enterprise test management, testing multiple code configurations, covered by analysis (CBA), probe points, change-based testing, and analytics.

Vector Software's award-winning VectorCAST embedded software testing platform is a family of products that automates testing activities across the software development lifecycle and supports C, C++, and Ada. The VectorCAST/C++ and VectorCAST/Ada applications are used for Unit and Integration Testing, while VectorCAST/QA is used for System Test Automation and Code Coverage Analysis.

[See also "VectorCast for Deos", AUJ 38-3, p. 119. —sparre]

Possibility of Janus/Ada for Linux

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 30 Apr 2018 21:32:01 -0500
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

> Could RR Software make a crowd-funding/pre-pay deal on a Linux version of Janus/Ada? How many licenses would you need to sell, to be ready to commit to deliver a Linux version of Janus/Ada?

Probably not that many (surely more than 20, probably less than 100), but I'm pretty booked until Ada 2020 is in the bag (summer 2019). So I wouldn't want to promise something that I couldn't complete. [I've thought about this project in the past, it probably wouldn't be too hard of a conversion -- we had SCO Unix compiler back in the day, so the front-end has needed case-insensitive file support.]

[See also "State of the Compiler Market", AUJ 38-2, p. 75. —sparre]

GNAT Pro Common Code Generator

From: AdaCore Press Center
Date: Fri May 4 2018
Subject: GNAT Pro Common Code Generator User's Guide Supplement 19.0w documentation
URL: http://docs.adacore.com/live/wave/gnat_ccg/html/gnatccg_ug/gnat_ccg/gnat_ccg.html

[...]

GNAT Pro Common Code Generator (also known as GNAT Pro CCG) is a compiler based on the GNAT Pro technology that takes a subset of Ada source code as input and generates corresponding C source code with the same semantics, suitable for compilation by any target C compiler. In other words, a subset of C is used as a high-level and portable assembly to compile Ada source code.

[...]

You need to run using gprbuild and specify the special "c" target, either via the --target command line switch, or via the Target project file attribute:

```
$ gprbuild -p --target=c -Pmy_project
```

or:

```
project My_Project is
- for Target use "c";
- [...]
```

[...]

Here is a list of constructs supported and not supported by this technology:

- Support for constructs that do not require runtime support:

  o packages (including child, separate, and generic packages)

  o subprograms (including separate, generic, nested, and overloaded subprograms)

  o most Ada types and subtypes, including:

  * scalar types (integer, enumeration)

  * floating point types

  * fixed point types

  * access types

  * constrained and unconstrained arrays

  * Note that support for multidimensional unconstrained arrays requires a C99 compatible C compiler. If the target C compiler only supports earlier versions of the C standard, then only one-dimensional unconstrained arrays can be used.

  * Array indices have to be within the bounds of the size_t C type, in other words you cannot use e.g. array indices of more than 2**31-1 on 32-bit targets.

  * record types

  * tagged types: Objects of tagged and class-wide types may be declared. Dispatching and class-wide subprograms are supported (including the object.operation notation).

  * private types

  * limited support for dynamically sized record types: records with a single (last) field whose size depends on a discriminant. Note that this feature builds on top of the C idiom "field[1];" where field is actually a larger array (allocated explicitly with sufficient memory). This idiom was replaced by flexible array declaration: "field[];" in C99.

- Support for a minimal standard library only
- delay until statement
- Representation clauses that map to C bitfields only (records containing integer types only, of 1 to 64 bits)
- Unsupported record representation clauses will generate a warning (unsupported representation clause, assuming confirming) and the generated code will ignore the unsupported representation clause, assuming the clause is simply confirming the default layout. This assumption (and this warning) need to be verified manually.
- Support for packed arrays (e.g. arrays of booleans)
- Support for assertions/preconditions/postconditions
- Raising an exception (explicitly or implicitly via a runtime check or an assertion failure) is mapped to a call to a last chance handler subprogram. See Exception Handling for more details.
- Support for most runtime checks: range, access, index, divide-by-zero checks.
- Support for 'Size is implemented via the C sizeof() built-in.
- Support for 'Valid on floating point requires the C99 isfinite() function.
- No support for overflow checks
- No support for subprograms in generic packages instantiated inside a subprogram, or for runtime pre/postconditions on subprograms in generic packages.
- No support for the following constructs requiring runtime:

  o tasking

  o controlled types

  o interface types

  o exception handling

  o storage pools

  o functions returning unconstrained arrays or dynamically sized records

- No support for 'Image on floating point and fixed point types, only scalars
- No support for the following attributes:

  o Alignment

  o Component_Size

  o Rounding

  o Bit

  o Bit_Position

  o First_Bit

  o Last_Bit

  o Position

o Constrained

o Mechanism_Code

o Null_Parameter

o Passed_By_Reference

- No support for assembly insertion. If you need to include assembly code, you can do so by putting the assembly in a separate assembly file.

Ada and Operating Systems

Ravenports: AdaYaml

From: John Marino <dragonlace.cla@marino.st>
Date: Sun, 15 Apr 2018 14:15:33 -0700
Subject: Re: ANN: AdaYaml 0.3.0
Newsgroups: comp.lang.ada

I've added AdaYaml to Ravenports.

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

[2]: http://www.ravenports.com/catalog/bucket_1F/AdaYaml/standard/

As far as I can tell, it's the first time AdaYaml is available in a package repository. Thanks for the nice project.

[See also "AdaYaml" earlier in this issue. —sparre]

MacOS: GCC

From: Simon Wright <simon@pushface.org>
Date: Sat, 12 May 2018 16:05:09 +0100
Subject: ANN: GCC 8.1.0 for macOS
Newsgroups: comp.lang.ada

GCC 8.1.0 for macOS El Capitan and High Sierra, built for C, C++, Ada, Fortran, Objective-C, and Objective-C++, and including

- ASIS, AUnit, GDB from GNAT GPL 2017
- Gprbuild, GNATColl, and XML/Ada from Github repositories, see the README

is available at

https://sourceforge.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/8.1.0/native-2017/

[See also "Mac OS X: GCC", AUJ 38-2, p. 77. —sparre]

References to Publications

ARG Progress Report

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sun, 1 Apr 2018 22:32:47 -0500
Subject: Interesting article on ARG work
Newsgroups: comp.lang.ada

I just ran across an article about the ARG's recent work. Read it at: http://www.ada-auth.org/ai-files/grab_bag/AAN-41.html

Ada in Context

Ada 2020 Parallelism Constructs

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sun, 1 Apr 2018 22:35:10 -0500
Subject: Re: Large number of tasks slows down my program (using debian) - any fix?
Newsgroups: comp.lang.ada

> [...]

The plan is for parallel blocks and loops, and a parallel reduction operation. There's more, but how much will get actually finished by early next year is obviously a question. The things I mentioned the are most likely.

Conditional Compilation and Program Variants

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sun, 15 Apr 2018 16:07:45 +0300
Subject: Ada conditional compilation and program variants
Newsgroups: comp.lang.ada

> [...]

Starting from the joke, the thread passed on to consider several existing compiler features, and serious suggestions, for controlling which parts of the Ada source code are used by the compiler, and for what purposes, including

- the proposal for "ghost code" in Ada 2020
- other extensions to the "contract" features of Ada
- the inconvenience, in current Ada, of having to comment-in and comment-out debugging code, when the static-Boolean-conditional method does not work (e.g. to include or exclude a "with" clause)
- Janus/Ada conditional compilation, with the "@" marker
- the risk that some combinations of the conditions for conditional compilation result in a variant of the source code that is syntactically or semantically illegal, with an example from Randy for "@"
- Dmitry's suggestions for changes to the Janus/Ada "@" to reduce that risk by requiring "@" to control only whole syntactic constructs (i.e. to work on the grammar level rather than the lexical level)
- Dan'l's suggestions for using GLR parsing, or related fork/merge parsing methods, to ensure that all possible variants of the source code are syntactically legal, with a reference to an implementation for C-with-preprocessor that has successfully done this for the Linux kernel code (with an interesting connection to Dmitry's suggestion in that one essential part of this implementation is "ifdef-hoisting" where the ifdef scopes are expanded to control whole syntactic constructs).

It seems to me that a common question for the above points is how to manage "variants" of an Ada program (and, as Dan'l commented, "aspects" might be included, because they have a similar need to be separated from the rest of the program). Variants may be necessary to support different target systems, or different compilers, or just to choose which optional features (such as debugging or state-consistency checking) should be included in a particular build of the program.

The traditional solution is to use a preprocessor to conditionally select or transform the source code before the compiler sees it. This however means that the basic source code is not Ada, but Ada-with-preprocessor, and it also leads to differences between implementations, for example Janus/Ada with "@" versus the AdaCore/GNAT preprocessor commands. And of course all the old arguments against macro-based, text-oriented preprocessors are still valid.

Barring preprocessors, a practical solution, which I often use and which I believe is widely used, is to isolate the features and variant code into their own packages (or separate subprograms), to provide variant bodies for those packages (e.g. for different targets, or to include or omit debugging), and to guide the Ada compiler to select the desired bodies through some kind of search path (e.g. ADA_INCLUDE_PATH for GNAT, or the GNAT project files for gprbuild). This often works well, but also often leads to some amount of duplicated invariant code in the various package bodies, because isolating exactly and only the variant code into packages would create a mess of very many small packages, possibly conflicting with the logical modular structure of the program. Furthermore, just as for the preprocessor method, this variant-bodies method is not standardized and is therefore supported in different ways by different IDEs and compilers.

It further seems to me that this thread has identified some desirable requirements for an Ada conditional-compilation / variant-support feature, including:

1. It should be defined in the Ada standard, to ensure portability across compilers and to make it easier for IDEs to support it by e.g. hiding or colorizing inactive parts of the source code.

2. It should be a part of the Ada language (the grammar), and should control whole syntactic constructs, unlike macro-based, text-oriented conditional compilation directives or preprocessors.
3. It should allow implementations to use GLR/FMLR-like parsing and analysis methods that can process all possible variants "at once" and check their legality as far as possible.

The variant-bodies method provides all of these, to some extent. It follows point 1 because all the source code is standard Ada. It follows point 2 because the boundaries of the variant code are package or subprogram boundaries. It follows point 3 because the static conformance of each body variant with all package declarations (and thus with any variant body of any other package) can be ensured by compiling each body variant separately, without having to build the whole program for all possible combinations of variant bodies. (This assumes that only bodies have variants, and declarations are invariant; unfortunately, declarations often need variants, too.)

On the other hand, the variant-bodies method fails to provide some aspects of the above requirements: it fails on point 1 because the method of choosing a particular variant of a body is not standardized; it supports point 2 weakly, because it is limited to variants that are complete bodies, and usually requires a dedicated source-code file for each variant; and for point 3 it forces implementations of the "all-variants-at-once" processors to use compiler/builder-specific methods for finding the possible variants of each body.

At present, I don't have a suggestion for a better method (than the variant-bodies method), but I think it could be a fruitful extension to Ada, especially if it could support all three uses: variants (different implementations of a non-optional feature of the program), optional features, and aspects (by which I mean the centralized specification, at one point in the program, of distributed actions taken at several, appropriate points in the program).

In fact, I have one idea that could be part of this Ada extension: "package definitions". A package definition would be a new kind of compilable unit (but generating no code), usually in its own source-code file, which would bear a similar relation to a package declaration as a package declaration currently bears to the package body:

- A package declaration declares the things that the corresponding package body is required to implement.
- Analogously, a package definition would define, or specify, the things that the corresponding package declaration is required to declare.

However, this definition would be on a less specific level (more "generic") than the actual declarations, and would therefore allow different package declarations (variants) to conform to the same package definition.

For example, the definition of package A could require package A to declare a type T that is a discrete type, but the definition might not require anything more of T. Thus, one variant of package A might declare T as an integer range, while another could define T as an enumeration.

The package definition would be, for example:

```
package definition A is
   type T is (<>);
   procedure F (Item : in T);
end A;
```

and a possible conforming package declaration would be:

```
package A is
   type T is (X, Y, Z);
   type S is new String;
   procedure Foo (Item : in T);
   procedure Bar (Item : in out T);
end A;
```

The Ada RM is in fact full of package definitions, because the declarations of the predefined, standard packages shown in the RM contain text like

```
subtype Any_Priority is Integer range
/implementation-defined/;
```

which a package definition would state as (for example):

```
subtype Any_Priority is Integer range <>;
```

Package definitions would be optional, but if a package definition is present then the package declaration must conform to it. This would allow the "variant bodies" method to be extended to allow also variant declarations, as long as the package definitions are invariant.

An interesting question is this: if package declaration or body B uses package A, and package A has a definition, how far can a compiler check that B uses A in legal ways if the compiler is allowed to look only at the definition of A, but not at the (or a) declaration of A?

If the compiler could check legality using only the package definitions of server packages, it would make it easier to ensure that all program variants are legal, by two separate and non-combinatorial steps:

- checking each variant package declaration against the (invariant) package definition, and
- checking each variant package body against the (invariant) package definitions of all other packages (those used by this body).

If checking the legality of how B uses A requires looking at the declaration of A (and not only at the definition of A), there is again a risk of combinatorial explosion in checking the legality of all (complete) program variants when there are variant package declarations.

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon, 07 May 2018 10:41:48 +0200
Subject: Re: Ada conditional compilation and program variants
Newsgroups: comp.lang.ada

> [...]

The only counter-examples I've been able to come up with so far are run-time errors, such as going beyond the range of a numerical type.

What about enumeration types? Should it be allowed to declare names for some, but not all values of an enumeration type? One could of course declare a function returning a value of the type in the package definition, and say that it can be implemented in the package declaration as an enumeration value. - Can't do that for characters though, so maybe it is a bad idea.

Termination and Finalization of Library Level Tasks

From: Egil Harald Høvik <ehh.public@gmail.com>
Date: Sun, 15 Apr 2018 12:32:18 -0700
Subject: Re: Finalization of library level tasks
Newsgroups: comp.lang.ada

Dmitry A. Kazakov wrote:

> Terminate alternative is almost always useless because it cannot be mixed with "else" or "delay". If there were a way to check within the task if its completion has been requested

For library level tasks:

```
loop
   select
      accept Some_Rendezvous;
   or
      delay Some_Interval;
   end select;

   exit when not
      Ada.Task_Identification.Is_Callable
(Ada.Task_Identification.Environment_Task);
  end loop;
```

Map Iterators of the Trait Based Containers

From: Jere <jhb.chat@gmail.com>
Date: Sat, 5 May 2018 09:03:53 -0700
Subject: Re: Trait based containers
Newsgroups: comp.lang.ada

[...] I hope they revert the changes to Map iterators if they do add it to Ada202x. I'm not sure why they changed it to iterate over the keys instead as that seems like the least useful way to iterate over a map, and you could still access keys via the cursor. Now the less useful way is the default.

From: Emmanuel Briot <briot@adacore.com>
Date: Mon, 7 May 2018 00:02:34 -0700
Subject: Re: Trait based containers
Newsgroups: comp.lang.ada

> [...]

One of the main principle in the design of that library is that people could change most of the details. For instance, if you do not like the default iterator that returns a key, you can easily derive your own that returns an element instead.

Initially, I had wanted to return a tuple (key, element) directly. This is really what a cursor is. So instead of doing

```
for A of Map loop
```

you do

```
for C in Map.Iterate loop
```

and you get both key and element.

I prefer to return a key in the "for ... of ... loop" case, because with a map you can always go from key->element. The opposite might not be true.

Trait Based Containers

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 5 May 2018 09:43:39 +0100
Subject: Trait based containers
Newsgroups: comp.lang.ada

I thought this was being submitted for inclusion into Ada 202X?

What's happening with this?

From: Emmanuel Briot <briot@adacore.com>
Date: Mon, 7 May 2018 00:00:10 -0700
Subject: Re: Trait based containers
Newsgroups: comp.lang.ada

> [...]

I am the original developer for those experimental containers. This was mostly a way to explore the use of generics in various contexts, and they resulted in a few (I believe 3, though I forgot exactly) new AI submitted to the ARG. Those AI are not about that library itself, but about various improvements to the language to better support this kind of generic packages.

I don't think we ever wanted to integrate them into the language. Indeed, Ada already has its own set of containers, so it would be a hard sell to provide a second set of containers.

Instead, the intent was for AdaCore to base its implementation of the standard Ada containers on something like the traits containers, for better code sharing among other things, and maybe also to provide the traits containers as an independent library (I think this is better, as someone else will come with a better implementation/design at some point, so better if it isn't hard-coded in the RM).

Then the SPARK people took interest in those as well, since, as opposed to the Ada containers, they provide a good basis for SPARK-compatible containers. Claire Dross did a huge amount of work to make the containers provable (not proven, this isn't the same thing).

There were quite a few limitations in Ada that made those containers somewhat hard to use for final users. In particular, instantiating the containers is somewhat tricky, especially if you want to control all the details (if you use one of the high-level packages, this is very similar to instantiating Ada's own containers).

Then I left AdaCore...

At this point, I don't think there's anyone working on those containers from the pure Ada library standpoint. I hope that the SPARK people still have an interest in a subset of them, but I must admit I have no internal knowledge of this.

In my new company, we use a lot of similar constructs, and in the process discovered a lot of compiler bugs, mostly related to the use of constructs like expression functions, for ... of loops, and others.

It would be nice if there was still interest in that library. The code is all accessible on GitHub, and I believe the license is GPL.

Object'Image

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Tue, 8 May 2018 22:24:58 +0200
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

[...]

Anyway, [Object'Image] is just a minuscule improvement to writing, with no new feature and no benefit in reading.

Use the good old' T'image(v), and don't worry about versions.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Wed, 9 May 2018 11:27:50 +0200
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

> Anyway, it's just a minuscule improvement to writing, with no new feature and no benefit in reading.

I disagree. It is more than that

I affects 'with' statements sometimes.

And it makes the code look better/easier to read

Below an uncompiled example of

- old ways needs more 'withs' - which may affect elaboration order

- makes the code harder to read

```
package Core_Types is
   type Handling_Unit_System_Status_Type
    is (Auto, Semi_Auto, Out_Of_System);
    type Assignment_Result_Type is
           (Success, Failure, Severe_Failure);
    type Assignment_Identity_Type is
           1 .. 99_999;
end Core_Types;

with core_types;
package Trp_Assignments is
  type Assignment_type is record
     Assignment_Identity :
       core_types.Assignment_Identity_Type ;
     Assignment_Result :
       core_types.Assignment_Result_Type;
     Handling_Unit_System_Status:
        core_types.
         Handling_Unit_System_Status_Type;
  end record;

with core_types; -- ONLY NEEDED FOR
                 -- do_stuff_old_way
 package body Transport_Handler is

   procedure do_stuff_old_way(Asm :
Trp_Assignments.Assignment_type) is
   begin
    -- NEED THE 'with core_types'
    log("id" & core_types.
         Assignment_Identity_Type'Image(
              Asm.Assignment_Identity)
        & " " &"result " & core_types.
         Assignment_Result_Type'image(
              Asm.Assignment_Result)
        & " " & "status " & core_types.
         Handling_Unit_System_
            Status_Type'Image(
        Asm.Handling_Unit_System_Status));
   end do_stuff_old_Way;

   procedure do_stuff_new_way(Asm :
       Trp_Assignments.Assignment_type) is
   begin
    -- DOES NOT NEED THE 'with
    -- core_types'
    log("id" &Asm.Assignment_Identity'Image
        & " " &"result " &
        Asm.Assignment_Result'Image
        & " " &"status " & Asm.
       Handling_Unit_System_Status'Image);
   end do_stuff_new_Way;

end Trp_Assignments;
```

> Use the good old' T'image(v), and don't worry about versions

Not when can be avoided - for me anyway

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Wed, 9 May 2018 11:56:46 +0200
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

> I affects 'with' statements sometimes.

That's a good point, however if you have a variable of a type declared in another package, I think it is extremely unlikely that you don't need to "with" the package for some other reason (like, in your example, simply assigning a value to the variable).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 9 May 2018 16:53:51 -0500
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

> [...]

It should be noted that one can use Obj'Image exclusively once it is available, because the prefix can always be qualified if necessary and that adds exactly one character to the attribute reference (and it's a lot shorter if qualification isn't needed).

My plan (once I'm using a Janus/Ada that supports it consistently, probably a few years still even though it is supported in the leading edge version now) is to completely forget the subtype version exists and use the object version consistently.

(Note that J-P was one of the few people that didn't support this extension, so it's fair for him to be consistent - but he was wrong then and he's wrong now. ;-)

Worst Features of Ada

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 9 May 2018 17:56:04 -0500
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

[...] that led to several of the worst features of Ada. And those had no impact on the uptake (or lack thereof) of Ada. I surely hope that we've (the Ada community) learned better by now.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 10 May 2018 00:55:33 +0100
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

> [...]

What are they?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 10 May 2018 17:10:26 -0500
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

> [...]

All of the stuff about anonymous access types along with some other details.

Compiler Warnings about Run-time Range Checks

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 11 May 2018 16:45:06 -0500
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

> [...]

Also note that in the common special case of loop iteration, no checks are needed at all, either at the generation of the index or its use:

```
for I in Arr'Range loop
   ... Arr(I) ...
end loop;
```

I cannot be outside of its range by construction, so no checks needed there other than the usual loop termination check, and that being the case, no checks are needed on the array indexing, either.

For many of the Ada checks, the existence of a check that can fail indicates poorly written Ada code and/or a potential bug. I'm working toward having the compiler (optionally) identify these during compilation, so that one can improve the code to eliminate the danger *before* testing.

Target Specific Bodies and Version Control

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 11 May 2018 17:12:20 -0500
Subject: Re: AI12-0218: What is the portable representation clause for processing IETF packets on little-endian machines?
Newsgroups: comp.lang.ada

> [...]

The best way to do this in Ada is with different package bodies (and sometimes specs) for each target. That's how Janus/Ada is designed, and it works great. I know the GNAT project system even has facilities to make this happen automatically (by selecting the unit to compile based on a version id).

The problem is that existing version control systems cannot handle such designs properly. That was definitely true in the late 1980's (so I designed a wrapper around our version control to deal with this), and I haven't seen any that deal with it properly to date. (The main issue being that when a bug is fixed in one such package, you want a notification to check if the same is needed for the other versions. No version control that I'm aware of can handle this - they all seem focused on merging development for a single end-product.)

Shortcomings in version control are way outside of anything that the language can control. And using a sub-optimal design because ancillary tools are broken seems to be letting the tail wag the dog.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Sat, 12 May 2018 12:33:18 +0200
Subject: Re: AI12-0218: What is the portable representation clause for processing IETF packets on little-endian machines?
Newsgroups: comp.lang.ada

> The best way to do this in Ada is with different package bodies (and sometimes specs) for each target.

We do this too.

> The problem is that existing version control systems cannot handle such designs properly.

To get around that we have subdirectories matching the target:

- w32_x86

- lnx_x64

- aix_ppc

Each of these directories would contain the native body and spec for the platform. Each directory and file is version_controlled.

The build system (first homebrew, now GPR) only includes the directory matching the correct target. So when building on AIX, the compiler never sees the directories lnx_x64 and w32_x86.

Each platform tests on an environment variable we set that has one of the above 3 values.

So in essence we limit the files the compiler sees to the ones belonging to the current platform.

From: Simon Wright <simon@pushface.org>
Date: Sat, 12 May 2018 14:08:04 +0100
Subject: Re: AI12-0218: What is the portable representation clause for processing IETF packets on little-endian machines?
Newsgroups: comp.lang.ada

> [...]

I have to say that my experience at Cortex GNAT RTS[1] is similar to Randy's.

At the moment I have 6 usable branches; the gcc7* and gcc8* ones are live:

- gcc6 (FSF GCC 6, GNAT GPL 2016)

- gcc7 (FSF GCC 7)

- gcc-finalization (FSF GCC 7 + finalization)

- gcc8 (FSF GCC 8)

- gcc-finalization (FSF GCC 8 + finalization)

- gnat-gpl-2017

The reason for the distinction is that the interface between the compiler and the RTS changes; between gcc7 and gcc8 10 files changed, and between gcc8 and gcc8-finalization 30 files changed (not including tests).

The way I've "managed" this is to choose a branch to develop a change on, implement the changes, commit, then checkout the other branch that the development applies to and cherry-pick the commit(s) (i.e. the changes) from the first branch. This is error-prone, to say the least. It'd be quite hard to do at all on my previously preferred DVCS, Mercurial.

Your remarks about using directories for this are very interesting, I'll be looking into them. I suspect the problem of "change a spec; apply matching changes to all the variant package bodies" will remain, though.

[1] https://github.com/simonjwright/cortex-gnat-rts

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Sat, 12 May 2018 16:21:16 +0200
Subject: Re: AI12-0218: What is the portable representation clause for processing IETF packets on little-endian machines?
Newsgroups: comp.lang.ada

> Your remarks about using directories for this are very interesting, I'll be looking into them.

It is a way of getting all files into version handling.

> I suspect the problem of "change a spec; apply matching changes to all the variant package bodies" will remain, though.

Unfortunately yes.

Agile Programming

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 14 May 2018 17:21:26 -0500
Subject: Re: How to get Ada to ?cross the chasm??
Newsgroups: comp.lang.ada

[...]

Agile programming itself is simply a style where one uses a relatively short development cycle with the intent to have a usable/testable version of the end product as early and as continuously as possible. This allows feedback from the customer and from testing the product itself.

I've *always* programmed in an agile manner, long before anyone decided to get rich giving it a name and pushing high-priced seminars. IMHO, the lower-case "agile" is the only sane way to develop software.

Imagine building an Ada compiler using a waterfall model. You'd spend a year creating design documents. And for what: a well-designed Ada package specification is itself a fine design document. I spend time (usually outside of the office) thinking about the design of new features, packages, and so forth, but I rarely write any of that down until I actually outline the package specifications involved. (Sometimes these I'll write down an outline, but that's mainly because of the reality of my work these days (and also advancing age!), when I might very well go several weeks between starting/designing a project and having enough time to actually complete the implementation. I won't remember the design details without some notes.)

I also have a meta-rule that I try not to write more code at a time than I can write/compile/test in a single day. (Not always practical, but I much prefer that.) I try to break down each task into subtasks that can be completed in a day and preferably tested by itself. The basic idea is to always have a working compiler (that's critically important if a customer needs an immediate fix -- doesn't happen very often but you can be sure that it will happen when the compiler isn't usable :-). Whenever one has a working compiler, it's possible to find regressions (a very common problem in compiler work), bugs, and some omissions via the test suites (which are trivial to run; these days I have them run by a batch file called "X"; can't get more trivial than that).

Ada works great for agile programming, as Niklas said, because the compiler helps make sure that you've made all of the changes needed as part of a new feature. It might make refactoring a bit harder, but that's actually OK -- a lot of refactoring is just churn - it might make the code look a bit better, but it really doesn't help the ultimate goal (getting a working product). So it helps put a brake on the desire to "simplify" everything; rather, one only does it when there is really an important reason (usually because code needs to be generalized to support a new requirement).

Revolutionize your software verification

Efficiency, Automation, Reliability

RAPITA Systems Ltd
A DANLAW Company

Unit testing · System testing · Coverage analysis · Timing analysis
V&V services · Multicore timing services · DO-178C training
Ada · C · C++

www.rapitasystems.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.

2018

July 03-06 — 30th **Euromicro Conference on Real-Time Systems** (ECRTS'2018), Barcelona, Spain. Topics include: all aspects of real-time systems, such as scheduling design and analysis, real-time operating systems, hypervisors and middlewares, virtualization and timing isolation, mixed-criticality design & assurance, worst-case execution time analysis, modelling and/or formal methods, industrial use-cases and real-time applications, tools, compilers and benchmarks for embedded systems, etc. Event includes: Worst-Case Execution Time analysis (WCET), Workshop on Analysis Tools and methodologies for Embedded and Real-time Systems (WATERS).

- July 03 — 9th **International Real-Time Scheduling Open Problems Seminar** (RTSOPS'2018). Topics include: single-, multi- and many-core scheduling; new models for real-time systems; scheduling in cyber-physical systems; mixed-criticality scheduling; interactions between WCET (worst-case execution time) analysis and scheduling; etc.

☺ July 09-10 — **Workshop: Konstruktion von SafeWare - Construction of SafeWare** (KSW'2018), Karlsruhe, Germany. Co-organized by Ada-Deutschland.

July 14-17 — 30th **International Conference on Computer-Aided Verification** (CAV'2018), Oxford, UK. Topics include: theory and practice of computer-aided formal analysis methods for hardware and software systems, algorithms and tools for verifying models and implementations, specifications and correctness criteria for programs and systems, deductive verification using proof assistants, program analysis and software verification, formal methods for cyber-physical systems, verification methods for parallel and concurrent systems, testing and run-time analysis based on verification technology, applications and case studies in verification and synthesis, verification in industrial practice, formal models and methods for security, etc.

July 15-17 — 22nd **International Symposium on Formal Methods** (FM'2018), Oxford, UK. Topics include: formal methods for the engineering of computer-based systems and software; such as industrial applications of formal methods; experience with formal methods in industry; tool usage reports; advances in automated verification, model-checking, and testing with formal methods; tools integration; environments for formal methods; development processes with formal methods; usage guidelines for formal methods; etc.

July 16-20 — 18th IEEE **International Conference on Software Quality, Reliability and Security** (QRS'2018), Lisbon, Portugal. Topics include: reliability, security, availability, and safety of software systems; software testing, verification and validation; program debugging and comprehension; fault tolerance for software reliability improvement; modeling, prediction, simulation, and evaluation; metrics, measurements, and analysis; software vulnerabilities; formal methods; benchmark, tools, and empirical studies; etc.

☺ July16-22 — 32nd **European Conference on Object-Oriented Programming** (ECOOP'2018), Amsterdam, The Netherlands.

July 23-27 — 42nd Annual IEEE **Conference on Computer Software and Applications** (COMPSAC'2018), Tokyo, Japan.

July 25-28 — 37th ACM **Symposium on Principles of Distributed Computing** (PODC'2018), Royal Holloway, University of London, UK.

☺ August 28-31 24th IEEE **International Conference on Embedded and Real-Time Computing Systems and Applications** (RTCSA'2018), Hokkaido, Japan. Topics include: real-time operating systems, real-time scheduling, timing analysis, programming languages and run-time systems, middleware systems, design and analysis tools, multi-core embedded systems, operating systems and scheduling, embedded software and compilers, fault tolerance and security, embedded systems and design methods for cyber-physical systems, applications and case studies of IoT and CPS, cyber-physical co-design, etc.

August 29-31 44th **Euromicro Conference on Software Engineering and Advanced Applications** (SEAA'2018), Prague, Czech Republic. Topics include: information technology for software-intensive systems; conference tracks on DSLs and Model-Based Development (DSLMBD), Software Process and Product Improvement (SPPI), etc.; tentative special sessions on Cyber-Physical Systems (CPS), Software Engineering and Technical Debt (SEaTeD), Monitoring Large-Scale Software Systems (MoLS), etc.

August 29-31 12th **International Symposium on Theoretical Aspects of Software Engineering** (TASE'2018), Guangzhou, China. Topics include: theoretical aspects of software engineering, such as abstract interpretation, component-based software engineering, cyber-physical systems, distributed and concurrent systems, embedded and real-time systems, formal verification and program semantics, integration of formal methods, language design, model checking and theorem proving, model-driven engineering, object-oriented systems, program analysis, reverse engineering and software maintenance, run-time verification and monitoring, software architectures and design, software testing and quality assurance, software safety, security and reliability, specification and verification, type systems, tools exploiting theoretical results, etc.

September 03-07 33rd IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2018), Montpellier, France. Topics include: foundations, techniques, and tools for automating the analysis, design, implementation, testing, and maintenance of large software systems; maintenance and evolution; model-driven development; reverse engineering and re-engineering; specification languages; software analysis; software architecture and design; software product line engineering; software security and trust; testing, verification, and validation; etc.

September 04-06 4th **Symposium on Dependable Software Engineering: Theories, Tools and Applications** (SETTA'2018), Beijing, China. Topics include: formalisms for modeling, design and implementation; model checking, theorem proving, and decision procedures; scalable approaches to formal system analysis; integration of formal methods into software engineering practice; contract-based engineering of components, systems, and systems of systems; formal and engineering aspects of software evolution and maintenance; parallel and multicore programming; embedded, real-time, hybrid, and cyber-physical systems; mixed-critical applications and systems; safety, reliability, robustness, and fault-tolerance; applications and industrial experience reports; tool integration; etc.

September 05-07 14th **International Conference on integrated Formal Methods** (iFM'2018), Maynooth, Ireland. Topics include: hybrid approaches to formal modeling and analysis; i.e., the combination of (formal and semi-formal) methods for system development, regarding both modeling and analysis, and covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice.

September 09-12 **Federated Conference on Computer Science and Information Systems** (FedCSIS'2018), Poznan, Poland. Event includes: 3rd International Workshop on Language Technologies and Applications (LTA), Joint 38th IEEE Software Engineering Workshop and 5th International Workshop on Cyber-Physical Systems (SEW & IWCPS), etc.

Sep 30 - Oct 05 **Embedded Systems Week 2018** (ESWEEK'2018), Torino, Italy. Topics include: all aspects of embedded systems and software. Includes CASES'2018 (International Conference on Compilers, Architectures, and Synthesis for Embedded Systems), CODES+ISSS'2018 (International Conference on Hardware/Software Co-design and System Synthesis), EMSOFT'2018 (International Conference on Embedded Software).

Sep 30 - Oct 05 **International Conference on Compilers, Architecture, and Synthesis for Embedded Systems** (CASES'2018). Topics include: the latest advances in compilers and architectures for high-performance, low-power embedded systems; leading edge research in embedded processor, memory, interconnect, storage architectures and related compiler techniques targeting performance, power, security, reliability, predictability issues for both traditional and emerging application domains; innovative papers

addressing design, synthesis & optimization challenges in heterogeneous, accelerator-rich architectures.

Sep 30 - Oct 05 ACM SIGBED **International Conference on Embedded Software** (EMSOFT'2018). Topics include: the science, engineering, and technology of embedded software development; research in the design and analysis of software that interacts with physical processes; results on cyber-physical systems, which compose computation, networking, and physical dynamics.

☺ October 10-12 26th **International Conference on Real-Time Networks and Systems** (RTNS'2018), Poitiers, France. Topics include: real-time applications design and evaluation (automotive, avionics, space, railways, telecommunications, process control, multimedia), real-time aspects of emerging smart systems (cyber-physical systems and emerging applications, ...), real-time system design and analysis (real-time tasks modeling, task/message scheduling, mixed-criticality systems, Worst-Case Execution Time (WCET) analysis, ...), software technologies for real-time systems (model-driven engineering, programming languages, compilers, WCET-aware compilation and parallelization strategies, middleware, Real-time Operating Systems (RTOS), hypervisors), formal specification and verification, real-time distributed systems (fault tolerance, task/messages allocation, ...), etc.

October 11-12 12th ACM/IEEE **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2018), Oulu, Finland. Topics include: the strengths and weaknesses of software engineering technologies and methods from a strong empirical viewpoint, including quantitative, qualitative, and mixed studies; case studies, action research, and field studies; replication of empirical studies and families of studies; mining software engineering repositories; empirically-based decision making; assessing the benefits/costs associated with using certain development technologies; industrial experience, software project experience, and knowledge management; software technology transfer to industry; etc. Deadline for submissions: July 1, 2018 (Emerging Results, Vision papers), July 20, 2018 (industrial papers, posters), August 10, 2018 (Journal-First track).

October 15-18 29th IEEE **International Symposium on Software Reliability Engineering** (ISSRE'2018), Memphis, Tennessee, USA. Topics include: innovative techniques and tools for assessing, predicting, and improving the reliability, safety, and security of software products; reliability, availability and safety of software systems; validation and verification; faults, errors, failures, defects, bugs; software quality and productivity; software security; dependability, survivability, fault tolerance and resilience of software systems; systems (hardware + software) reliability engineering; open source software reliability engineering; supporting tools and automation; industry best practices; virtualization and software reliability; empirical studies of any of the above topics; software standards; etc. Deadline for submissions: July 1, 2018 (industry papers, industry abstracts/presentations, tutorials), July 21, 2018 (workshop papers, fast abstracts, doctoral symposium).

October 16-19 15th **International Colloquium on Theoretical Aspects of Computing** (ICTAC'2018), Stellenbosch, South Africa. Topics include: semantics of programming languages; theories of concurrency; theories of distributed computing; models of objects and components; timed, hybrid, embedded and cyber-physical systems; static analysis; software verification; software testing; model checking and automated theorem proving; verified software, formalized programming theory; etc.

☺ November 04-09 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2018), Boston, Massachusetts, USA. Topics include: all aspects of software construction, at the intersection of programming, languages, and software engineering. Events include: ACM SIGAda's HILT workshop (High Integrity Language Technology for Cybersecurity in Real-Time and Safety-Critical Systems). Deadline for submissions: July 1, 2018 (DLS - Dynamic Languages Symposium), July 6, 2018 (GPCE - Generative Programming: Concepts & Experiences, SLE - Software Language Engineering), July 20, 2018 (Doctoral Symposium), July 27, 2018 (Student Research Competition), August 17, 2018 (workshop papers), September 22, 2018 (posters), end of September 2018 (Student Volunteers applications).

Nov 05-06 11th ACM SIGPLAN **International Conference on Software Language Engineering** (SLE'2018). Topics include: areas ranging from theoretical and conceptual contributions, to tools, techniques, and frameworks in the domain of software language engineering; generic aspects of software languages development rather than aspects of engineering a specific language; software language design and implementation; software language validation; software language integration and composition; software language maintenance (software language reuse, language evolution, language families and

variability); domain-specific approaches for any aspects of SLE (design, implementation, validation, maintenance); empirical evaluation and experience reports of language engineering tools (user studies evaluating usability, performance benchmarks, industrial applications); etc. Deadline for submissions: July 6, 2018 (papers), August 31, 2018 (artifacts).

November 04-09 12th **Joint European Meeting** of the **Software Engineering Conference** and the ACM SIGSOFT **Symposium on the Foundations of Software Engineering** (ESEC/FSE'2018), Orlando, Florida, USA. Topics include: architecture and design; components, services, and middleware; debugging; dependability, safety, and reliability; development tools and environments; distributed, parallel, and concurrent software; education; embedded and real-time software; empirical software engineering; formal methods, including languages, methods, and tools; model-driven software engineering; processes and workflows; program analysis; program comprehension and visualization; refactoring; reverse engineering; safety-critical systems; scientific computing; security and privacy; software economics and metrics; software evolution and maintenance; software modularity; software product lines; software reuse; testing; traceability; etc. Deadline for submissions: July 22, 2018 (JPF workshop), July 27, 2018 (EnSEmble, A-TEST, SWAN, WASPI workshops), August 31, 2018 (NL4SE workshop).

☺ November 05-06 ACM SIGAda's **High Integrity Language Technology International Workshop on Languages and Tools for Ensuring Cyber-Resilience in Critical Software-Intensive Systems** (HILT'2018), Boston, Massachusetts, USA. Co-located with SPLASH 2018. Organized by ACM SIGAda. Topics include: language features that can be used to build security and/or safety into software-intensive systems; extending contract-based programming to specifying security resistance and resilience properties as well as safety and/or correctness properties; modeling and/or programming language features and analysis techniques that aid in code analysis and verification and that increase the level of abstraction and expressiveness; language features that support continuous requirements maturation to support evolving needs, particularly in cyber-physical systems, while ensuring that security and safety properties are preserved; etc. Deadline for submissions: July 31, 2018 (papers, extended abstracts).

November 10-13 18th **International Conference on Runtime Verification** (RV'2018), Limassol, Cyprus. Topics include: monitoring and analysis of the runtime behaviour of software and hardware systems. Application areas include cyber-physical systems, safety/mission-critical systems, enterprise and systems software, autonomous and reactive control systems, health management and diagnosis systems, and system security and privacy.

November 28-30 19th **International Conference on Product-Focused Software Process Improvement** (PROFES'2018), Wolfsburg, Germany. Topics include: experiences, ideas, innovations, as well as concerns related to professional software development and process improvement driven by product and service quality needs. Deadline for submissions: August 5, 2018 (short papers, tools, demos, posters).

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

☺ December 11-14 39th IEEE **Real-Time Systems Symposium** (RTSS'2018), Nashville, Tennesse, USA. Topics include: all aspects of real-time systems, including theory, design, analysis, implementation, evaluation, and experience. Deadline for submissions: September 19, 2018 (workshop papers).

2019

January 08-11 31st **Conference on Software Engineering Education and Training** (CSEET'2019), Grand Wailea, Maui, USA. Topics include: curriculum development; empirical studies; personal or institutional experience; team development; software assurance, quality, and reliability education; methodological aspects of software engineering education; global and distributed software development; open source in education; cooperation between industry and academia; etc.

January 15-18 11th **Software Quality Days Conference** (SWQD'2019), Vienna, Austria. Topics include: improvement of software development methods and processes; testing and quality assurance of software and software-intensive systems; domain specific quality issues such as embedded, medical, automotive systems; novel trends in software quality; etc.

☺ April 01-04 **International Conference on the Art, Science, and Engineering of Programming** (Programming'2019), Genova, Italy. Topics include: programming practice and experience; general-purpose programming; distributed systems programming; parallel and multi-core programming; security programming; interpreters, virtual machines and compilers; modularity and separation of concerns;

model-based development; testing and debugging; program verification; programming education; programming environments; etc. Deadline for submissions: July 1, 2018 (workshops, deadline 1), September 1, 2018 (workshops, deadline 2).

April 6-12 22nd **European Joint Conferences on Theory and Practice of Software** (ETAPS'2019), Prague, Czech Republic. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), POST (Principles of Security and Trust), TACAS (Tools and Algorithms for the Construction and Analysis of Systems).

April 08-12 34th ACM **Symposium on Applied Computing** (SAC'2019), Limassol, Cyprus. Deadline for submissions: September 10, 2018 (papers), September 20, 2018 (tutorials).

♦ June 10-14 24th Ada-Europe **24th International Conference on Reliable Software Technologies** (Ada-Europe 2019), Warsaw, Poland. Sponsored by Ada-Europe, in cooperation (pending) with ACM SIGAda, SIGBED, SIGPLAN, and the Ada Resource Association (ARA).

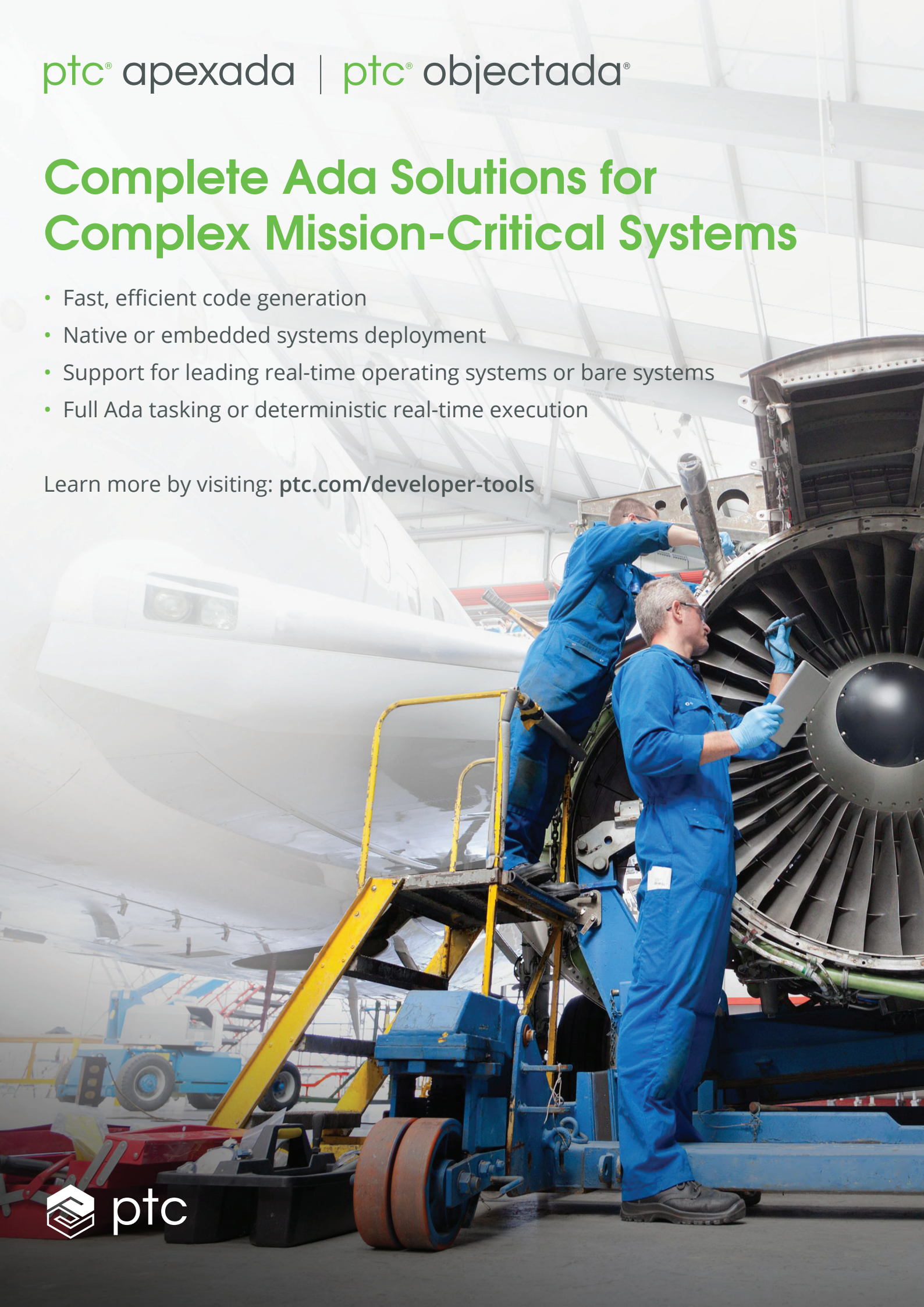
December 10 Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

ptc® apexada | ptc® objectada®

Complete Ada Solutions for Complex Mission-Critical Systems

- Fast, efficient code generation
- Native or embedded systems deployment
- Support for leading real-time operating systems or bare systems
- Full Ada tasking or deterministic real-time execution

Learn more by visiting: **ptc.com/developer-tools**



ptc

Call for papers and extended abstracts

HILT 2018

Workshop on *Languages and Tools for Ensuring Cyber-Resilience in Critical Software-Intensive Systems*

As part of SPLASH 2018, November 5 & 6, 2018, Boston, MA, USA
Sponsored by ACM SIGAda

The *High Integrity Language Technology* (HILT) 2018 Workshop is focused on the cyber-resilience needs of critical software systems, where such a system must be trusted to maintain a continual delivery of services, as well as ensuring safety in its operations. Such needs have common goals and shared strategies, tools, and techniques, recognizing the multiple interactions between security and safety.

We encourage papers and extended abstracts relating to:

- Language features that can be used to build security and/or safety into software-intensive systems;
- Approaches to apply effectively the emerging technologies of AI and Machine Learning in critical software systems;
- Mechanisms that can be used to understand, certify, and manage systems that are "data driven," relying on "soft code," where control flow and algorithms are expressed using data rather than "hard code" expressed directly in programming languages;
- Extending contract-based programming to specifying security resistance and resilience properties as well as safety and/or correctness properties;
- Strategies to minimize risk when applying complex software requirements to cyber-physical systems;
- Modeling and/or programming language features and analysis techniques that aid in code analysis and verification and that increase the level of abstraction and expressiveness;
- Language features that support continuous requirements maturation to support evolving needs, particularly in cyber-physical systems, while ensuring that security and safety properties are preserved.

This workshop is designed as a forum for communities of researchers and practitioners from academic, industrial, and governmental settings, to come together, share experiences, and forge partnerships focused on integrating and deploying tool and language combinations to address the challenges of building cyber-resilient software-intensive systems. The workshop will be a combination of presentations and panel discussions, with one or more invited speakers.

Attendees wishing to present at the workshop should prepare full papers (approx. 6-8 pages), or extended abstracts (approx. 2-4 pages) for their proposed presentations, and the workshop program committee will select presentations and organize them into sessions. Other interested participants are welcome to register for the HILT 2018 Workshop as part of their SPLASH 2018 registration.

July 31: Papers or Extended abstracts due;
Aug 31: Notification of submissions accepted for presentation
Sep 30: Final submissions due
Nov 5&6: Workshop as part of SPLASH 2018

Please submit papers and extended abstracts, by July 31, 2018, by following the link from: http://sigada.org/conf/hilt2018

Workshop Co-Chairs

- Bill Bail, MITRE
- Tucker Taft, AdaCore, Inc

Organizing Committee

- Dirk Craeynest, ACM SIGAda International Representative, KU Leuven
- Drew Hamilton, Chair, ACM SIGAda, Mississippi State University, CCI
- Clyde Roby, Secretary-Treasurer, ACM SIGAda, Institute for Defense Analyses
- Alok Srivastava, Editor, ACM Ada Letters, Engility Corp.
- Ricky E. Sward, Past Chair, ACM SIGAda, MITRE

URLs

- SPLASH 2018: http://www.splashcon.org
- HILT 2018: http://sigada.org/conf/hilt2018
- ACM SIGAda: http://sigada.org

Ada-Europe

24th International Conference on Reliable Software Technologies

10-14 June 2019, Warsaw, Poland

Conference & Program Chair

Tullio Vardanega
University of Padova, Italy
tullio.vardanega@unipd.it

Educational Tutorial & Workshop Chair

Dene Brown
SysAda Ltd, UK
dene.brown@sysada.co.uk

Industrial Chair

Maurizio Martignano
Spazio IT, Italy
maurizio.martignano@spazioit.com

Exhibition & Sponsorship Chair

Ahlan Marriott
White Elephant GmbH, Switzerland
software@white-elephant.ch

Publicity Chair

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

Local Chair

Maciej Sobczak
GE Aviation – EDC Warsaw, Poland
maciej.sobczak@ge.com

General Information

Ada-Europe is pleased to announce that its 24th **International Conference on Reliable Software Technologies** (Ada-Europe 2019) will take place in Warsaw, Poland. The conference schedule at its fullest includes a three-day technical program and vendor exhibition from Tuesday to Thursday, and parallel tutorials and workshops on Monday and Friday.

Schedule

14 January 2019	Submission of papers, industrial presentation outlines, tutorial and workshop proposals
1 March 2019	Notification of acceptance to all authors
16 March 2019	Camera-ready version of papers required
30 April 2019	Industrial presentations, tutorial and workshop material required

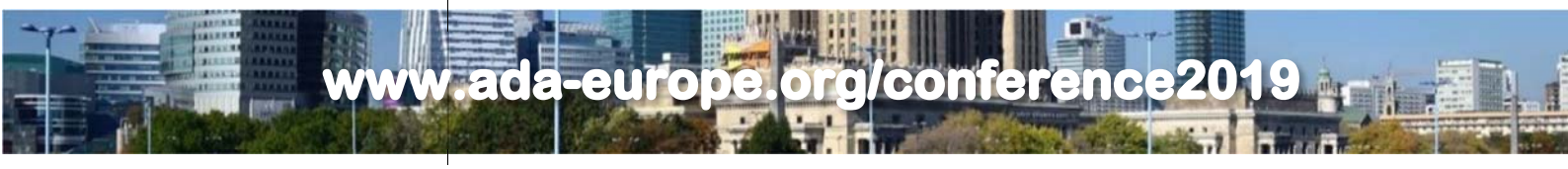
Topics

The conference is a leading international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a challenging variety of application domains. The program will allow ample time for keynotes, Q&A sessions and discussions, and social events. Participants include practitioners and researchers from industry, academia and government organizations active in the promotion and development of reliable software technologies.

The topics of interest for the conference include but are not limited to:

- Design and Implementation of Real-Time and Embedded Systems,
- Design and Implementation of Mixed-Criticality Systems,
- Theory and Practice of High-Integrity Systems,
- Software Architectures for Reliable Systems,
- Methods and Techniques for Quality Software Development and Maintenance,
- Ada Language and Technologies,
- Mainstream and Emerging Applications with Reliability Requirements,
- Experience Reports on Reliable System Development,
- Experiences with Ada.

Refer to the conference website for the full list of topics.



Call for Regular Papers

The regular papers submitted to the conference must be original and shall undergo anonymous peer review. The corresponding authors shall submit their work by 14 January 2019. Such submissions shall be in PDF only and up to 16 LNCS-style pages in length. The authors shall use the EasyChair submission service at *https://easychair.org/conferences/?conf=adaeurope2019*.

The International Conference on Reliable Software Technologies is listed in the principal citation databases, including DBLP, Scopus, Web of Science, and Google Scholar.

Proceedings

The conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the conference, both online and in print. The authors of accepted regular papers shall prepare camera-ready submissions in full conformance with the LNCS style, strictly by 16 March 2019. For format and style guidelines, the authors should refer to *http://www.springer.de/comp/lncs/authors.html*. Failure to comply and to register at least one author for the conference by that date will prevent the paper from appearing in the proceedings.

Call for Industrial Presentations

The conference seeks industrial presentations that deliver insightful information value but may not sustain the strictness of the review process required for regular papers. The authors of industrial presentations shall submit their proposals, of at least 1 page in length, by 14 January 2019, strictly in PDF, using the submission service at *https://easychair.org/conferences/?conf=adaeurope2019*.

The Industrial Committee will review the submissions anonymously and make recommendations for acceptance. The authors of accepted contributions shall be requested to submit a 2-page abstract by 30 April 2019, for inclusion in the conference booklet, and be invited to deliver a 20-minute talk at the conference. These authors will also be required to expand their contributions into articles for publication in the *Ada User* Journal (*http://www.ada-europe.org/auj/*), as part of 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 Educational Tutorials

The conference is seeking tutorials in the form of educational seminars including hands-on or practical demonstrations. Proposed tutorials can be from any part of the reliable software domain, they may be purely academic or from an industrial base making use of tools used in current software development environments. We are also interested in contemporary software topics, such as IoT and artificial intelligence and their application to reliability and safety.

Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half day or full day), and the intended level of the tutorial (introductory, intermediate, or advanced). All proposals should be submitted by e-mail to the Educational Tutorial Chair.

The authors of accepted full-day tutorials will receive a complimentary conference registration. For half-day tutorials, this benefit is halved. The Ada User Journal will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

Workshops on themes that fall within the conference scope may be proposed. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference week. Workshop proposals should be submitted to the Workshop Chair. The workshop organizer shall also commit to producing the proceedings of the event, for publication in the *Ada User Journal*.

Call for Exhibitors

The commercial exhibition will span the core days of the main conference. Vendors and providers of software products and services should contact the Exhibition Chair for information and for allowing suitable planning of the exhibition space and time.

Special Registration Fees

Contributors to the conference and all students will enjoy reduced registration fees.

Venue

The conference will take place in Warsaw, the capital of Poland, at the facilities of the Engineering Design Center (EDC), a partnership between General Electric (GE) Poland and the Institute of Aviation, one of the largest engineering institutions in Europe.

Proceedings of the 19th International Real-Time Ada Workshop

18-20 April 2018
Benicàssim, Spain

Contents

Program Committee

Mario Aldea Rivas, John Barnes, Ben Brosgol, Alan Burns, Michael González Harbour, Kristoffer Gregertsen, José Javier Gutiérrez, Stephen Michell, Brad Moore (Program Chair), Luís Miguel Pinho, Juan Antonio de la Puente, Jorge Real (Workshop Chair), Pat Rogers, José F. Ruiz, Sergio Sáez, Joyce Tokar, Tullio Vardanega, Andy Wellings and Rod White.

Workshop Participants

Mario Aldea Rivas, Spain
Alan Burns, UK
Alfons Crespo, Spain
Jorge Garrido, Spain
Michael González Harbour, Spain
Francisco Gómez Molinero, Spain
Brad Moore, Canada
Kristoffer Nyborg Gregertsen, Norway
Luis Miguel Pinho, Portugal
Eduardo Quiñones, BSC, Spain
Juan Antonio de la Puente, Spain
Jorge Real, Spain
Sara Royuela, Spain
Sergio Sáez, Spain
Tullio Vardanega, Italy
Andy Wellings, UK
Juan Zamorano, Spain

Sponsors

Summary of the 19th International Real-Time Ada Workshop

Brad Moore, **Program Chair**
General Dynamics Mission Systems-Canada, Canada; email: brad.moore@gd-ms.ca
Jorge Real, **Workshop Chair**
Universitat Politècnica de València, Spain; email: jorge@disca.upv.es

Abstract

Since the late Eighties, the International Real-Time Ada Workshop series has provided a forum for identifying issues with real-time system support in Ada and for exploring possible approaches and solutions, and has attracted participation from key members of research, user, and implementor communities worldwide. Recent International Real-Time Ada workshops have contributed to the Ada 2005 and Ada 2012 standards, especially with respect to tasking features, the real-time and high integrity systems annexes, and the standardisation of the Ravenscar Tasking Profile.

The 19th International Real-Time Ada Workshop was held in Benicàssim, near Valencia, Spain. The main focus of the workshop was to build upon what is considered a key strength and a core feature of Ada, which is tasking. Ada has always been a premier language for tasking and real-time support. The proposals and discussions of the workshop were all about extending these capabilities to fit better with a wider variety of target system architectures. In one sense, the notion of tasking is expanded to fit larger scale systems including multicore and heterogeneous systems with the addition of new parallelism features. At the same time, the notion of tasking is minimalized and restricted to fit better on highly resource-constrained targets where previously multi-tasking was not considered a good fit. Another related topic was about improving determinism for Ada tasks that have more critical needs for timing and accuracy, in terms of time triggered task scheduling and with regard to clocks and synchronisation. For all these features, consideration was given for use with the Ravenscar Profile, which is a flexible yet minimalist runtime amenable to analysis and schedulability. Finally, the discussion also covered a new variant of the EDF task dispatching policy that uses the Deadline Floor Protocol for controlling access to protected objects, which is conceptually a simpler approach.

The workshop had great success in refining existing proposals and in identifying important new ones. The delegates thoroughly enjoyed their time in the scenic setting, and are very grateful to the organizers for all their efforts.

1 Introduction

The 19th International Real Time Ada Workshop (IRTAW) was held in Benicàssim, Spain on April 18-20, 2018. The following are the main set of topics were discussed at the workshop:

- Parallel Programming issues;
- Multiprocessor Locking Issues;
- Language Profiles;
- Time Triggered Scheduling Issues;
- Deadline Floor Protocol Issues;
- Language Issues; and
- Clock Issues

The workshop was attended by participants from Canada, Italy, Norway, Portugal, Spain, and the United Kingdom. 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, also published in this issue.

In this report, Sections 2 to 8 provide the high-level session summaries of the workshop. Section 9 provides a list of AI's to be produced by the workshop. The plan for next meeting is considered in section 10. Finally, section 11 concludes with the closing of IRTAW-19.

2 Session: Parallel Programming

This session provided an overview of the new parallelism features being proposed for Ada 202x, and then considered how the OpenMP tasking model could be combined with the Ada tasklet model, as well as provided the motivation for doing so. Miguel Pinho started the workshop off by first presenting the new parallelism features being proposed for Ada 202x. An important point to mention is that because potentially-blocking operations during parallelism are not allowed in the current proposal, many complex problematic issues were avoided. A concern was raised about the proposed Associative aspect, and the suggestion was made to drop the aspect from the proposal, since the compiler is generally unable to determine the associativity of a function. Another suggestion was to disallow the exception keyword directly in a parallel block statement, due to confusion about where the handler was being applied. It was also discussed whether fine tuning controls of the parallelism should be added to

Figure 1: IRTAW-19 Participants: From Left to Right: Francisco Gómez-Molinero, Sergio Sáez, Alan Burns, Andy Wellings, Jorge Real, Michael González Harbour, Jorge Garrido, Mario Aldea-Rivas, Tullio Vardanega, Brad Moore, Kristoffer Gregertsen, Luís Miguel Pinho, Juan Zamorano, Alfons Crespo, Juan Antonio de la Puente, missing from photo: Sara Royuela, Eduardo Quiñones

the language. The consensus was the eventually such tuning would be needed, but it is too premature to try to define for Ada 202x.

Sara Royuela, and Eduardo Quiñones proceeded with a presentation based on the position paper of Pinho, Royuela, and Quiñones entitled: "Combining the tasklet model with OpenMP" [**?**]. The motivation for integrating the OpenMP tasking model with Ada's tasklet model, is that OpenMP provides a broader set of capabilities than the standard is currently proposed to provide. In particular, OpenMP supports unstructured parallelism which is useful for certain problems. Furthermore, tools could be provided that automatically generate dependencies for parallelism, and there is a lot of experience behind the OpenMP standard that would appeal to those familiar with the standard.

Brad Moore then presented the barrier-like abstraction he calls a Synchronous Signal, described in his position paper entitled: "Synchronous Signals: An Abstraction for Interleaving Sequential and Parallel Code" [1], which is useful when transitioning between parallel and sequential sections of code. The synchronous signal abstraction works like a synchronous barrier, except that it involves less synchronisation, which can improve performance.

Kristoffer Gregertsen then made an industrial presentation about his experience working on various projects involving parallelism. The main point of his presentation is that serious parallelism is more commonly applied to heterogeneous processor environments, rather than multicore environments. It was mentioned that OpenMP now provides support in this area, which is another reason to consider integrating Ada with OpenMP.

3 Session: Multiprocessor Locking

Jorge Garrido provided a presentation on multiprocessor locking proposals currently being considered, as described in the position paper of Garrido, Zamorano, and A. de la Puente entitled "On protocols for accessing protected objects on multiprocessors" [2]. In particular MrsP and MSRP were compared. While it was felt by the group that it would be premature to look at standardising these protocols, it was noted that most multiprocessor locking protocols, including these two, specify that locks should be obtained in FIFO order, but Ada currently does not specify any ordering for obtaining access to protected objects. It was felt that this capability was needed and was important enough to request inclusion in the Ada 202x standard. This resulted in AI12-0276-1 (Admission Policy Defined for Acquiring a Protected Object Resource) being created and submitted to the ARG for consideration.

4 Session: Profiles

Mario Aldea presented a proposal to support Ada tasking on devices with very limited memory, based on the position paper of Aldea-Rivas and Pérez-Tijero entitled "Proposal for a new Ada profile for small microcontrollers" [3]. The problem to be overcome is that the memory needed for stacks in a set of tasks collectively consumes too much memory resources. The main idea behind the presentation was to invoke sufficient restrictions so that all tasks could share the same stack space. For example, tasks are not allowed to have local state. A challenge remains in how to specify restrictions in a manner that the compiler can verify if they are being honoured. Part of the discussion was about whether Ada's Non-Preemptive scheduling can support this idea, or if it can be modified to better support this. After reflecting on this, during the wrap up session at the end of the workshop, it was decided that for Non-Preemptive scheduling, Ada currently does not define sufficient preemption points. To provide better determinism, the preemption points should be defined in terms of initiating potentially blocking operations rather than initiating blocking operations. An AI was created to deal with this problem, and

subsequently submitted to the ARG for consideration for Ada 202x.

Tullio Vardanega then made a presentation discussing ongoing work associated with the position paper of Carletto and Vardanega entitled "Ravenscar-EDF: Further Results from Improved Comparative Benchmarking" [4] to review and provide better comparison data between EDF scheduling versus Fixed Priority scheduling. The experimentation attempts to achieve maximum fairness by involving the use of a Ravenscar Fixed Priority scheduler for the Leon processor, against a modified version of this runtime which replaces the Fixed Priority components of this runtime with EDF components. Early results confirm some of the held beliefs in comparing EDF-FPS, but other findings contrast with earlier results.

5 Session: Time Triggered Scheduling

Jorge Real presented the proposal of Real, Sáez, and Crespo entitled "Ravenscar Support for Time-Triggered Scheduling" [5] which describes an implementation for Time-Triggered scheduling for use with the Ravenscar Profile. The main idea of the proposal is to allow a programmer to create a time triggered cyclic plan of slots of various types and lengths of time where each slot is associated with a task or allows other priority based schedulers to execute priority based tasks. The tasks of the TT scheduler need to execute at top priority in the system. The proposal is flexible to handle many different types of plans. Questions remain about whether this should eventually be standardized, or instead treated as a framework utility that could be applied to a partition.

Another outcome of this discussion is that it was mentioned that the source code behind this presentation is available on git-hub. It was suggested that contributions such as this should be available in a common place, or organized in a package hierarchy along with contributions from other people, such as Brad Moore's Paraffin source, or Kristoffer Gregertsen's clock and timer related libraries. It was mutually agreed that a peer-reviewed repository under package XAda would be good way to tie these together and bring the community to a common place, as well as encourage people in the community to contribute their own ideas.

6 Session: Deadline Floor Protocol

Alan Burns presented a proposal for refining the integration of the Deadline Floor Protocol into Ada. The Deadline Floor Protocol offers the key benefits of the SRP protocol but is conceptually simpler. The work is of importance also because the associated AI, (AI12-0230-1) is one of the AI's selected for inclusion in the Ada 202x standard, provided that the details can be worked out in time. Alan has since progressed and updated the AI and submitted to the ARG for consideration.

7 Session: Ada Issues for Ada 202x

Tullio Vardanega presented the list of AI's being considered for inclusion in Ada 202x, that are of interest to the real-time community. It was mentioned that the AIs related to parallelism are some of the higher priority items being considered for standardisation, but in addition, three other AI's were mentioned to be of interest:

- AI12-0139-1 Thread Safe language-defined units
- AI12-0230-1 Deadline Floor Protocol
- AI12-0234-1 Compare and Swap for Atomic Objects

Brad Moore then went on to describe some possible directions to consider for AI12-0234. After presenting, it was agreed that a library of intrinsic calls should be provided that can be mapped to hardware instructions for operations such as compare and swap and atomic increment. In addition, the group unanimously agreed that pragma CPU should be specifiable for a protected object, which allows an implementation to provide a simpler lock-free implementation for a protected object if all tasks that use the protected object are assigned to the same CPU.

8 Session: Clock Issues

Kristoffer Gregertsen presented issues related to clocks and timers in Ada, as described in his position paper entitled "Clock support in Ada" [6]. Kristoffer noted that Ada appears to be falling behind other languages with regard to capabilities in this area. Other languages uses an object oriented approach, which Kristoffers feels would also be beneficial for Ada. Various issues were raised, including the need for better clock synchronisation, and availability of wall clock support for Ravenscar applications. It was also suggested that this could be another case for contributing to the XAda package repository that was endorsed by the group in the Time-Triggered Scheduling session.

9 Real-Time AIs to be considered for inclusion in Ada 202x

The IRTAW 19 workshop did have many good discussions about various real-time issues, which resulted in the need to create new AI's or progress existing ones so that they may be considered for inclusion in the Ada 202x standard. The list of AI's processed by the workshop are:

- AI12-0230-1 Deadline Floor Protocol
- AI12-0234-1 Compare and Swap for atomic objects
- AI12-0276-1 Admission Policy Defined for Acquiring a Protected Object Resource
- AI12-0279-1 Nonpreemptive Dispatching Needs more Dispatching Points
- AI12-028x-1 Need to Specify Atomic aspect on Generic Formal types
- AI12-028y-1 Allow CPU aspect to be specified on Protected Object Declarations

The last two AIs were originally considered to be part of AI12-0234-1, but the ARG asked these to be split into separate AIs to better separate the issues. They have not yet been assigned official AI numbers.

Alan Burns was assigned the Deadline Floor Protocol to work on. Brad Moore was assigned AI12-0234-1, AI12-0276-1, AI12-028x-1, and AI12-028y-1, and Tullio Vardanega was assigned AI12-0179-1.

10 Conclusions and Next IRTAW

Deadlines were set for finalization of session reports, production of final versions of the position papers, and writing of the AIs to be sent to ARG. There was general agreement about the need of future editions of IRTAW. The next edition, IRTAW 20 will be organized by Jorge Real returning to Benicàssim, Spain on April 20, in 2020. The group noted that it would be somehow appropriate to have the 20th IRTAW in 2020. Tullio Vardanega will head the role of Program Committee Chair.

11 Closing

There being no other pending issues, Brad Moore closed the session and the workshop. The workshop thanked the presence of first-time participants. All thanked Jorge Real for the splendid local arrangement in the scenic locale.

References

[1] L. M. Pinho, E. Quiñones (2018), *Position Paper: combining the tasklet model with OpenMP*, This Issue.

[2] B. Moore (2018), *Synchronous Signals: An Abstraction for Interleaving Sequential and Parallel Code*, This Issue.

[3] J. Garrido, J. Zamorano, J. A. de la Puente (2018), *On protocols for accessing protected objects on multiprocessors*, This Issue.

[4] M. Aldea-Rivas, H. Pérez-Tijero (2018), . *Proposal for a new Ada profile for small microcontrollers*, This Issue.

[5] P. Carletto, T. Vardanega (2018), *Ravenscar-EDF: Further Results from Improved Comparative Benchmarking*, This Issue.

[6] J. Real, S. Sáez, A. Crespo (2018), *Ravenscar Support for Time-Triggered Scheduling*, This Issue.

[7] K. Gregertsen (2018), *Position paper: Clock support in Ada*, This Issue.

Session Summary: Parallel Programming

Luis Miguel Pinho (Session Chair)
CISTER/ISEP, Portugal; email: lmp@isep.ipp.pt

Tullio Vardanega, (Rapporteur)
Università di Padova, Italy; email: tullio.vardanega@math.unipd.it

1 Introduction

The session centered on the discussion of three topics, each brought forward by a corresponding submission, backed by distinct participants in the Workshop:

1. Support for parallelism based on OpenMP, championed by Sara Royuela, Eduardo Quiñones, and Luis Miguel Pinho;
2. Synchronous signals, presented by Brad Moore;
3. Heterogeneous platforms, brought forward by Kristoffer Nyborg Gregertsen.

Before discussing the individual topics, Miguel and Brad provided a summary of the *status quo* for the model of parallelism that emanated from past editions of this Workshop and it is being considered by the Ada 202x language amendment process via the AIs listed in Table 1.

The general model that underpins those AIs is that the application code presents *potential opportunities for parallelism* (POPs), the compiler generates executable code for exploiting them, and the runtime schedules their execution. This model requires refining the notion of task, when it includes a parallel construct, to represent multiple logical threads of control that can proceed in parallel. Each such thread of control within a task is termed an *executor*, and the execution that it performs between synchronization points is termed a *tasklet*. When a task distributes its execution to a set of executors, it cannot proceed with its own execution until all the corresponding tasklets have completed. Those tasklets may synchronize by making protected operations, but cannot call blocking operations. The new Global (cf. AI12-0079-1) and Nonblocking (cf. AI12-0064-2) aspects may be used to facilitate the detection of such calls at compile time.

At the previous IRTAW this conceptual model was examined in the regard of the interaction of parallelism with tasking, determining the following semantics:

- Changing task attributes should be deferred until outside of the region of parallel execution;
- If parallel tasklets attempt to perform multiple changes on the same task attribute, one of them is selected arbitrarily to take effect;
- If multiple distinct operations are deferred during a parallel execution (such as, for example, a task attribute change and an exception), they should be applied as close as possible to what prescribed for them in sequential Ada;
- Whereas a per-CPU execution-time accounting would be desirable within parallel regions, the Workshop initially proposed a simpler model that only provides a per-task counter, which is updated at the end of the parallel region;
- Set_CPU and Get_CPU calls should be provided to specify CPUs where the tasklets of a task should execute.

2 Supporting Ada's parallelism with OpenMP

After recapping the current situation with the Ada 2020x AIs related with parallel programming, attention shifted to the first topic of discussion in the session agenda, i.e., the viability of supporting Ada's parallelism with OpenMP.

OpenMP is a widely used in the HPC domain and it is now also entering the embedded domain. The OpenMP solution has three parts: annotations (pragmas) applied to the user code to specify requirements for parallel execution; a compiler pass that acts on those annotations generating calls to the OpenMP runtime library; the runtime that manages parallel execution.

Guided in the discussion by Sara and Eduardo, the Workshop acknowledged that OpenMP's *tasking* model (with its unfortunate naming clash), when used in the classic fork-join semantics, maps quite well to Ada's *tasklet* model. In addition to that, however, OpenMP also allows for more flexible approaches to parallel programming, called "unstructured parallelism", which address scenarios where the progress of some execution within a parallel region is dependent on certain data-driven conditions to be met. OpenMP provides the pragma "task depend" to this end, to express out/in conditions for a particular flow of execution: for example, task A outputs an 'a' that task B uses, means that B cannot start before A completes. An intuition of how OpenMP's unstructured parallelism relates to Ada's strict fork-join model is depicted in Figure 1.

The Workshop acknowledged that, while very interesting, OpenMP's unstructured parallelism falls much outside of the scope of parallel programming in Ada 202x. With that

Table 1: Ada 202x AIs related with parallel programming

AI12-0119-1	Parallel blocks and loops	
AI12-0242-1	Reduce/Parallel_Reduce attributes	
AI12-0251-1	Explicit chunk index for parallel loops	dependent on AI12-0119-1
AI12-0251-2	Manual chunking operations	dependent on AI12-0119-1 and AI12-0266-1
AI12-0262-1	Map-reduce attributes	dependent on AI12-0242-1 and AI12-0212-1
AI12-0266-1	Parallel container iterators	dependent on AI12-0119-1
AI12-0267-1	Data race and blocking prevention	dependent on AI12-0064-2, AI12-0079-1, and AI12-0119-1

notion in mind, the discussion moved on to the issue of how OpenMP's runtime could be delegated Ada's tasklet semantics, while staying under control of the Ada runtime for scheduling. In particular, the Workshop discussed what should happen when an Ada task executing a parallel region within the OpenMP runtime would be preempted. The problem here is that, as OpenMP's threads of control have no notion of priority, as do OpenMP's tasks, the Ada binding should propagate priorities down to the underlying Operating System threads by means of (scarcely attractive) ad-hoc extensions to the OpenMP runtime. Alternative mapping solutions were explored, but none was found to be convincing, especially when interaction with POs and task attributes were to be contemplated.

The conclusion of this very interesting discussion was that more work is needed to understand, from an OpenMP perspective, how to mix concurrency and parallelism. The IRTAW group should seek the opportunity to discuss with the OpenMP community about this matter and strive to inject real-time concerns into it. There was consensus that this could be an interesting avenue to explore in the future.

3 Synchronous signals for parallel-sequential-parallel patterns of execution

Brad summarised the essence of his proposal on this point. In analogy with current Ada's Synchronous_Barriers library package added to the Real-Time Systems annex in 2012, Brad's idea was to provide a low-cost mechanism, named Synchronous_Signals, to allow sequential processing, within a concurrent unit, to interleave with parallel processing. One type of call to a Synchronous_Signals object would manage the transition from parallel to sequential code, and another call would manage the converse. In a region with

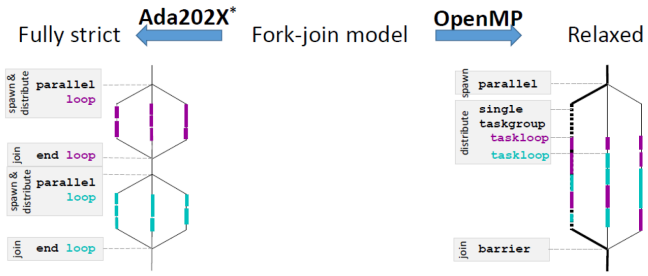


Figure 1: Ada's strict fork-join model vs. OpenMP's unstructured parallelism (picture taken from joint work by Sara, Miguel and Eduardo, presented at DATE 2018).

N threads of control, one designated caller of the former API would wait until all other $N-1$ had made the same call, and then be able to proceed alone sequentially. On its call to the latter API, all of the N threads of control would resume parallel execution. The intent of this mechanism is to minimize the amount of synchronization in a parallel application and make better use of the available CPUs.

The Workshop concurred that the "parallel-sequential-parallel" sequence of execution is a common pattern in parallelism, which justifies looking at this problem. On the merit of Brad's proposal, the sentiment of the Workshop was twofold. On the one hand, it was felt that the naming of the mechanism was not appropriate, as the notion of signal is much overloaded. On the other hand, as the intended application semantics can be implemented with POs, it was felt that a faster runtime implementation would be an insufficient argument to justify a new language object, which instead would if HW support existed for the intended semantics.

However, the opportunity of investigating support for this feature via libraries (*à-la* Paraffin) was deemed useful and interesting because it allows user exploration of possible uses of the feature, while avoiding the need to define application-level syntax for it and solve all problems of interaction of it with tasklets. To this end, the Workshop recommended that Brad's Paraffin library should be updated with this and the other recent features, be uploaded to a public repository, and referenced from an Ada-Europe's page for the benefit of the general public.

4 Ada and heterogeneous processor architectures

The session concluded with Kristoffer reporting on his recent work on heterogeneous processor architectures, and his reflection on the role that Ada could have in them.

The first observation was that massive SMP architectures are not common outside of HPC. Much more frequent, instead, are heterogeneous architectures, where OpenCL, OpenACC or CUDA are the dominant software platforms. To argue this point Kristoffer presented a robotic use-case application that an ESA-funded project currently is developing. The gist of the argument was that it would be nice, in that systems context, that an Ada application would be able to command and control accelerators. The value added of it would be the better safety and reliability associated with the quality of the language. This integration might be achieved by a binding library, which would wrap an accelerator kernel, suitably determined

by the corresponding cross-compiler, send it off to the target hardware, and eventually receive the result of that computation. Discussing possible implementation routes for this ambit, the Workshop concurred that there would be two principal options: to reuse existing support from e.g., OpenCL, but giving up on certification; or else to re-implement it all with Ada/SPARK. It is evident that the latter would be more attractive, but also much more costly to achieve.

In wrapping the session up on discussion of the rising importance of heterogeneous architectures, the Workshop noted that OpenMP's support for those types of systems increases the interest of exploring ways to integrate Ada with it.

Session Summary: Multiprocessor Locking

Juan A de la Puente (Session Chair)
Politecnica de Madrid (UPM), Spain; email: jpuete@dit.upm.es

Alan Burn (Rapporteur)
University of York, UK; email: alan.burns@york.ac.uk

There was one paper at the Workshop on this topic: On the Protocols for Accessing Protected Objects on Multiprocessors by Garrido, Zamorano and de la Puente.

Jorge Garrido gave a short presentation to introduce the issue of locking on multiprocessor architectures. He introduced the problems and briefly described a number of potential solutions. He noted that most of these solutions utilise spin locking and FIFO ordered queues.

With the current definition of Ada waiting to gain access to a PO (protected object) is not a blocking operation. On a single processor with priority ceiling locking any task calling a PO should find the PO unoccupied unless there is an error – so the issue of waiting to gain access does not apply. With multiprocessors this is clearly not the case. The reference manual (Section 9.5.1 Note 19) states:

"If two tasks both try to start a protected action on a protected object, and at most one is calling a protected function, then only one of the tasks can proceed. Although the other task cannot proceed, it is not considered blocked, and it might be consuming processing resources while it awaits its turn. There is no language-defined ordering or queuing presumed for tasks competing to start a protected action — on a multiprocessor such tasks might use busy-waiting . . . "

The meeting felt that this wording was no longer appropriate, there is a need to allow the program to specify a queuing discipline. Although it was felt premature to attempt to incorporate an existing complete protocol such as MrsP or MSRP into Ada, there was agreement that basic support for the multiprocessor case was now required.

The meeting agreed unanimously that:

- Ada should allow a queueing discipline to be defined;
- That this discipline should be based on spinning (busy-waiting); and
- That one allowed approach should be defined in Ada: FIFO spinning at the protected object's ceiling priority.

A possible name for this queuing discipline is Admittance_-Policy; a name for the one defined policy could be FIFO_-Spinning.

By spinning at the protected object's (PO's) ceiling this task prevents any other task on the same processor from calling this PO. The FIFO discipline is favoured as it helps to bound the maximum size of the queue and the time it takes for a lower priority task to gain access into the PO.

One example of using this facility is in a Ravenscar context. Here all tasks are statically allocated to processors. If each PO used by tasks on different processors is given a high ceiling priority then the blocking time for each task can be computed.

Session Summary: Profiles

Mario Aldea-Rivas, *Session Chair*
Universidad de Cantabria, Spain; email: mario.aldea@unican.es
Kristoffer Nyborg Gregertsen, *Rapporteur*
SINTEF Digital, Norway; email: kristoffer.gregertsen@sintef.no

1 Introduction

The goal of this session was to discuss different profile related aspects of Ada. Two position papers were discussed in the session: Tulio Vardanega presented work and performance testing of EDF for Ravenscar [3], while Mario Aldea-Rivas presented the work on a profile for very small run-times [1].

2 EDF with the Ravenscar profile

In the "Judgment day paper" of Giorgio Buttazzo, strong claims are presented about the efficiency of EDF scheduling compared to FPS [2]. The presented work of Paolo Carletto and Tulio Vardanega seeks to test these claims by implementing EDF scheduling on a Ravenscar run-time for LEON, and performing extensive testing to compare the EDF scheduling scheme with FPS for a large number of task sets.

To be as fair as possible, when comparing the two scheduling schemes, only the run-time environment is modified while the application code remains the same. There is no need to set a deadline explicitly as the run-time gets the deadline as a task attribute. No major changes were needed to implement EDF in the Ravenscar run-time environment, as the whole run-time kernel is at about 1000 statements. EDF was implemented with a simple queue, without queue insertion optimization. However, for smaller task sets (i.e. queue lengths) this insertion scheme is also optimal. If the task suspends itself, its deadline will be adjusted when it is resumed. For interrupt handling, if the first task on the queue handles an interrupt, the rest of the queue is frozen until the handler has completed.

The EDF and FPS run-time environments were tested on open-source version of the LEON emulator. About 5500 different tasks sets with differing numbers of tasks (up to 40) were explored. The system is overloaded at up to 130% for some of the task sets. In the "judgment day" paper it was claimed that the number of pre-emptions for EDF will decrease with an increasing number of tasks. This was not found by work associated with the position paper, which instead noted that the number pre-emptions are about the same as for FPS. It was noted that the other paper implemented EDF on a fixed priority system, which might have been a factor. Furthermore it does not provide details on the used tasks set, so it is not repeatable. The ongoing work associated with the presented paper does not yet show any obvious benefits of EDF compared against FPS.

Another issue with EDF is that if the system temporarily goes over 100% utilization, EDF has cascading loss of deadlines. EDF performs worse in overload situations, and it is hard to say which tasks suffers most in overload. The author of the "judgment day" paper has argued that tasks should be abandoned on deadline overrun as in ERIKA. It could however be an issue to identify overrun situations and know which task to abort. In a mixed priority system, how can one avoid the most critical tasks missing their deadlines? Alan Burns, Andy Wellings and Michael Gonzales-Harbour gave some options on how to handle this. It could for instance be an option to change scheduling to FPS in case of overruns. However, this would require a shadow-queue, which would double the overhead of queue-handling. Questions were raised on how to make a ready-queue that eases the switch, and how to identify the overrun situation. Execution-time monitoring or EDF with priority levels are also possible. However, Tullio argues that we need a mapping to priority as an invertible function.

Alan asked if it is mostly the theoretical behaviour, or if the implementation details contribute much to the performance results as well. Tullio replies that both the theoretical behaviour and the implementation contribute. The EDF implementation spends more execution-time on insertion and removal because the time type is larger in size than the priority type. Time is implemented as a record on the LEON, and is more complex than the short integer used for priorities.

The authors of the presented paper are now abandoning the LEON run-time environment and are moving to the ARM for more testing. The workshop participants looks forward to the results of this work.

3 Profile for small run-times

Mario Aldea-Rivas presented a proposal for a new profile for small microcontrollers, such as the Arduino Uno that uses the ATmega328P from Microchip (previously Atmel). This is a typical small and inexpensive microcontroller with 32 KiB of program memory and only 2048 Bytes of SRAM. Such small MCU's have a large market share, but are mostly dominated by the C programming language. There are some Ada toolchains for the AVR architecture such as AVR-Ada and AVR GNAT 2012 for Windows hosts. Common for all these are that they use zero footprint (ZFP) run-time environments that do not support tasking. There are Ravenscar run-time environments for small ARM Cortex M microcontrollers, and also for the Microchip/Atmel AVR32 UC3 microcontroller.

Also there is one for 8-bit AVR Ravenscar implemented by one of Miguel Pinho's students.

The problem when porting Ravenscar to small MCU's is the stack size. For example, the default stack size for tasks (4 KiB) is larger than the entire SRAM memory for many MCU's. The minimum stack size is in the range of 200B and 500B in FreeRTOS, and it also allows for stack sharing for so-called "one-shot" tasks. These are tasks that do not keep state on the stack between activations, and therefore need to store all their internal state in global variables. Preemptive stack sharing could need a big stack, depending on task priorities. Non-preemptive scheduling does not have this problem. The question is how to define a new profile with the necessary restrictions to support a shared stack, similar to how the Ravenscar Profile defines restrictions for a simpler tasking runtime?

The proposal is to define "one-shot tasks" where:

- Local state is stored in global variables that could be of any type;
- No other local state is shared between invocations;
- There is a dispatching point on every activation (entry or delay until);
- Non-preemptive scheduling is applied;
- No suspension between the preemption points; and
- No classical context switches are allowed where registers are stored on the stack etc.

There is a preliminary implementation of one-shot tasks in M2OS. Three boards are supported, Arduino Uno, Raspberry Pi, STM32F. A tool analyses the code (output from GNAT) and detects violation of the required pattern needed for the one-shot tasks. The program memory footprint is very low at about 4 KiB and the stack usage for the example programs is less than 100 Bytes.

Tullio Vardanega asked if it would not be just as easy to use a pre-defined task implementation for this, such as a real-time utility framework. Andy Wellings suggested for that case, one could allow tasking only in the package implementing this task with the restrictions needed for one-shot tasks. Michael Gonzáles Harbour noted that one also needs the ability to block on entry calls, in addition to the periodic blocking. Alan Burns reminded the group that profiles are collections of restrictions for tasking, and that it would not be easy to have a restriction that enforces the required task pattern. Jorge Real asked if it would be better to have a library package, providing access types for the activation object, global data, and the code to be executed. Juan Zamorano indicated for that case, one would need to specify the Nonblocking aspect for the formal parameters associated with the code passed to the library, as well as possibly the formal types if the library is a generic. Tullio added that it might be useful if there were a GUI or tool/library to ease the usage for developers. Sergio Sáez, who lectures about Arduino at the University of Valencia, mentioned that there they use state machines implemented for program logic, which is easier to understand than tasking. A question was raised about the use of class-wide types for this abstraction. The concern was whether the Nonblocking aspect could be proven in that case. It turns out this is likely not an issue, since the Nonblocking aspect is inherited for dispatching operations.

A conclusion of the discussion was that the pattern cannot easily be defined as a set of restrictions, however, the workshop is supportive of the work. It also needs to be evaluated if it is better to have a standard package with tasking implementing the pattern, instead of a set of restrictions possibly associated with a new profile.

The point was also raised that Ada's non-preemptive dispatching policy would be a good fit for this pattern, but that there are some issues with the definition of this policy that would be problematic. Specifically, there is a more general issue with the wording in the language standard to support non-preemptive scheduling for one-shot tasks. Section D.2.4 states that:

> For this policy, blocking or termination of a task, a delay_statement, a call to Yield_To_Higher, and a call to Yield_To_Same_Or_Higher or Yield are the only task dispatching points (see D.2.1).

The issue, as later stated by Michael González Harbour and captured in the subsequently produced AI, is that:

> In non-preemptive dispatching it is necessary to have controlled points in the code where preemption is enabled, as a mechanism to keep the non-preemptive sections short and reduce blocking time for higher priority tasks. Unlike the preemptive case, these preemption points need to be predictable, always under the programmer's control. The current wording in the standard mentions "blocking of a task" as a preemption point. This can happen for instance when calling a closed entry or suspension object. However, if the entry or suspension objects are open there is no preemption point, which introduces uncertainty in the fact that a preemption point is reached, and therefore on the blocking time for higher priority tasks.
>
> The solution to this unpredictability is to declare all potentially blocking operations to be preemption points. In this case all calls to an entry or wait operations on a suspension point are preemption points regardless of whether they are open or not. This change would make non-preemptive dispatching more predictable.

A new AI (AI12-0279-1) was subsequently created by Tullio Vardanega to address the issue raised by the IRTAW workshop to look at replacing the meaning of "blocking" to "potentially blocking".

References

[1] M. Aldea-Rivas and H. Pérez-Tijero (2018), *Proposal for a new Ada profile for small microcontrollers*, This issue.

[2] G. C. Buttazzo (2005), *Rate Monotonic vs. EDF: Judgment Day*, Real-Time Systems, 29(1):5–26.

[3] P. Carletto and T. Vardanega (2018), *Ravenscar-EDF : Further Results from Improved Comparative Benchmarking*, This issue.

Session Summary: Time Triggered Scheduling in Ravenscar

Jorge Real (Session Chair)
Universitat Politècnica de València, Spain; email: jorge@disca.upv.es
Brad Moore (Rapporteur)
General Dynamics Mission Systems-Canada, Canada; email: brad.moore@gd-ms.ca

1 Introduction

The goal of this session was to discuss a proposal and implementation of a TT (Time Triggered) scheduler suitable for use with the Ravenscar Profile. The proposal extends a previous model that was intended for use with full-Ada, which was integrated with a PB (Priority-Based), preemptive scheduler. Jorge Real presented the proposal, which was based on an IRTAW paper [**?**] contributed by three authors, Jorge Real, Sergio Sáez, and Alfons Crespo, all of whom were present for the discussion.

2 Time Triggered Scheduling

Jorge first described the main idea behind the proposal. The idea is to combine a TT scheduling solution for tasks executing at the highest priority level, and then use a PB scheduler for the rest of the system. The proposal provides flexibility in choosing a suitable PB scheduler. For example an EDF scheduler might be chosen.

The original proposal presented at the previous IRTAW workshop showed how such a solution could work in the context of full Ada. The discussion for this session was focused on how to apply that strategy to the Ravenscar Profile. The TT scheduler is associated with an execution plan, which consists of a sequence of non-overlapping time slots or windows where each window corresponds to a task in the system, and each time slot can have independent durations. The plan is a cyclic plan that repeats over and over while the system is running.

Jorge described 5 different types of time slots that could be used in a plan:

- Regular — considered to have sufficient duration for TT actions
- Optional — Similar to Regular, but can be omitted
- Continuation
- Mode Change
- Empty

For a Regular time slot, a processing overrun is treated as an error (A Program_Error would be raised as an exception by the scheduler). A task must also be ready to use its allocated Regular time slot in the plan, otherwise Program_Error would be raised.

An optional time slot however, is a time slot that may or may not be used, and it is not an error if the optional time slot is not taken if a task is not ready to execute when the slot is started. In this case, the slot is made available for use by tasks managed by the PB scheduler. An optional slot is useful for cases such as sporadic tasks or communication tasks when there is nothing to communicate.

For a Continuation slot, there is no overrun check. A hold/resume mechanism is applied such that if a task overruns a Continuation slot, it is held until another Continuation slot or Regular slot occurs in the plan, at which point the task is resumed. A Regular slot is always used as a terminal slot for a set of Continuation slots for a given task.

The hold/resume mechanism is not currently implemented in existing runtimes, so the runtime would need to be modified to support this capability. It was noted that Ada does define the package, Ada.Asynchronous_Task_Control, which could be used for this purpose, but the package was not available for the target platform of interest. Jorge explained that one point to note was that only the scheduler would be allowed to make calls to Hold/Resume. It is important that user tasks not be given this capability, otherwise it makes it very difficult to analyse and schedule a plan, since the scheduler cannot know when a task would decide to resume.

If a task finishes earlier than the time allotted to a time slot, then the remaining time in the slot is available for tasks running under the PB scheduler. A question was asked, what happens if a task finishes before its terminal slot is started? It was explained that a propagation flag is applied where the early termination is propagated to the next slot. Multiple propagations may occur until it reaches the terminal slot. Continuation slots are useful for breaking up long time triggered processing in such a manner that is transparent to the system.

An Empty slot is a slot in which no time triggered action is processed. This is useful for planning gaps in processing where PB tasks can be guaranteed to execute.

A Mode Change slot is also a slot where there is no time triggered action to be processed. This is useful for performing actions such as changing the time triggered plan to a new plan.

Issues and Comments

It was mentioned that a hold operation of a Continuation slot could be a problem if it occurs in the middle of a protected action. The proposed solution is to have the Ceiling Priority of the protected object set to Interrupt_Priority'Last, which implies that protected operations should be very short. The holding needs to be done at a higher priority than the scheduler, and every PO in the time triggered scheduler needs to have the Ceiling Priority set to Interrupt_Priority'Last. This is necessary because a Continuation being held in a protected object is eating into the next time slot.

Michael González Harbour asked if there should be checks for overruns in this case. He suggested that the Ceiling Priority should be System_Priority'Last, not Interrupt_Priority'Last. Jorge explained that the Time Triggered scheduler is running at Interrupt_Priority, which is why Interrupt_Priority'Last was chosen.

Alan Burns raised a question about what the effect would be of a timing event goes off during a Hold. Jorge answered that in the current implementation, Hold does not check anything, even if in a protected action, so the simplest way to deal with this is to have the scheduler run at interrupt priority.

Tullio Vardanega commented that stating protected objects must be short is difficult to enforce, since "short" is a qualitative term. Tullio then suggests that the advice to keep things short is generally understood, but then asks if the framework able to protect against long operations. He further claims that saying that the run time doesn't check is not good.

Sergio Sáez responded and said that they would look into this, but then asks the group about how this could be improved. He added that the amount stolen from the next slot needs to be bounded, and as long as that bound is not overrun, that might be the best that can be done.

Alan Burns commented that he thought a hold should be a deferred operation, until after a protected action completes. An overrun is thus a timing_error. The thought was that this would make the scheduler more robust. A question was asked on how to make the scheduler treat this as a deferred operation?. Alan explained that there could be two timer events. One that says the slot is finished, and another that says you had better be out by this time. The second timer event would get canceled if you exit your slot. Alan Burns explains this is needed for robustness.

It was suggested that a protocol could be devised, such that before going into a non-interruptable section of code, a check would be made to see that you have enough time to complete the code. The protocol would decide to finish early if there were doubts in there being enough time for a non-interruptable event.

The discussion returned to the topic of using the Hold and Continue facilities of Ada.Asynchronous_Task_Control. Sergio commented that the package cannot be used with the Ravenscar Profile because the profile explicitly states that the package is not allowed. Tullio Vardanega commented that there should be things that the scheduler is allowed to do, but that the application cannot. Ada doesn't currently have this notion.

Michael commented that a new policy could be devised that would allow whatever rules are needed.

Tullio asked, for safety critical systems, can the runtime be trusted, but not the user? This is a question that cannot be answered offhand, but deserves more thought and analysis.

Andy Wellings changes the topic towards distinguishing between masking interrupts. In particular, he was interested in the case of inhibiting all interrupts except the timer interrupt, as the problem seems to be more about inhibiting interrupts than priority control. He adds that some interrupts are non-maskable, and the model of priorities may be too simplistic for interrupt handling.

We then resume to Jorge's slides. Jorge presents various patterns for time triggered plans which demonstrates that the model is very flexible and can accommodate a wide variety of possible plans and situations.

With regard to Continuation propagatation, Michael asks, why not number the slots with an Id, and then wait for the Id, rather than propagate continuations until reaching a terminal slot? Jorge answered that they had considered that, but they chose the current approach, as they felt it was a better approach requiring only boolean flags, rather than slot identifiers.

Jorge then presented a task pattern including a non-TT part, scheduled by the PB scheduler, and an Optional slot in the TT plan (it could also be Regular). The task normally runs in competition with other PB tasks, but it has a part that runs in sync with the plan (e.g., for timely data exchange with the TT part). This pattern requires this mixed task to run at two priorities, which in Ravenscar can only be enforced from the runtime. This pattern effectively uses dynamic priorities in Ravenscar, but in a restricted manner:

- A task lowers its own priority (by calling Leave_TT_Level) and raises it again by calling Wait_For_Activation.
- Priority changes only between TT and base priority levels
- Conceptually, it is like a "ceiling" inherited while a task runs its TT part.

Michael asks if data can be shared between TT and non-TT tasks, or if such communication could potentially spoil the plan. In particular, he was concerned about checking that there is progress, and whether there it was possible if the system could get out of sync, and if so, what mechanisms were there to bring the system back into sync.

There was some discussion about whether this would be a system issue or an application issue.

Jorge explained that the full Ada version has different mechanisms for these problems, but Ravenscar is a different game.

He noted that the pattern could be changed to re-sync every time it executed at the TT level, making this more an application issue than a system issue.

Tullio thinks it is good that the solution was brought to Ravenscar. A disadvantage might be a loss of control, but if it is rare, perhaps it is not so bad.

Another issue raised was the concern that TT scheduling appears not to be as flexible as other schedulers in that it requires reserving priority'last, and therefore needs to run at higher priorities than other schedulers. Andy commented that nowhere else in Ada do we say things only work at a certain priority level. For this model, we cannot apply it down at lower levels. In other schedulers, you can have bands of scheduling, and you can define where things go. Andy adds, What happens if I run time slicing at Priority'First? You'll probably get Program_Error on first overrun.

Andy suggested, Instead of Round_Robin within priority, perhaps it should be called TT at top priority. Michaels suggestion was to instead consider it as a policy by itself, which doesn't work with the priority bands. Andy said he liked that idea a lot better.

Alan commented by saying that he didn't think its a fundamental problem. Its unfortunate, but thats life. There is a way of using time triggered, and it involves reserving a priority for the scheduling.

Tullio reemphasised that the other dispatching policies were meant to coexist, but then Jorge noted that non-preemptive dispatching is another scheduler has similar issues coexisting with other schedulers.

Tullio ponders whether this is a framework that can be applied to a partition, or if should it be considered to be a language element. He suggested that the IRTAW group should reflect on this question. If it is an application utility, you are free to use it in any way. If it is a language element, it is a bigger issue.

Alan then raised a concern about seemingly putting things into Ravenscar that are not part of Ravenscar, which is another question that is left for future consideration.

Towards An IRTAW Common Repository

Jorge mentions that the source code for the runtime can be found in github.

Andy comments that there should be an IRTAW website with links to projects such as this, Brad's Paraffin parallelism libraries, the clock utilities proposed by Kristoffer Gregertsen, and other contributions coming from IRTAW or the real time community. Everyone agrees.

Andy notes that the proposed packages of this presentation are child packages of Ada, but it is not currently part of the standard, which would make it more difficult for others to use. He noted that if the goal is to have others use this, it should be under a different parent package. Perhaps something like Ada_Extensions, or XAda. After some discussion, if was felt that the XAda top-level package name would be the better choice, for it would be easier to distinguish from the standard "Ada.*" files, that would start with "a-*.ad?" in GNAT. This way, extensions would be named "x-*.ad?". The top level XAda parent package could be simply defined as; package XAda is pragma Pure; end XAda;

The group reached a consensus that such a naming framework would be good for unifying various contributions from the open source community. The group further mentioned that contributions should be peer-reviewed to gain acceptance before being added to this repository. There should be an official web-page on Ada-Europe, or similar, where information about these extensions are available, for instance hosted on GitHub.

In Summary

Jorge then summarises his presentation. He admits he is not fond of relying on Interrupt_Priority in the model, but it did solve problems. Some other remaining issues and questions are;

- Reserving priority for Priority'Last for the TT level might be seen as an issue. However, this is needed to ensure that PB tasks do not interfere with the plan. This is inherent and it was noted as such. Having restrictions on how to use scheduling policies is natural and the application has to be prepared for that. Failing to comply with this restriction would be like failing to assign task priorities correctly.
- Reserving Interrupt_Priority'Last as the ceiling of POs used by sliced TT tasks (i.e., those that can be held at the end of continuation slots) might be seen as an issue. This is not inherent with the proposal, but just a mechanism to ensure that the scheduler is not going to hold a sliced task in the middle of a protected action. This raised concerns about the difficulty to enforce "short" protected actions and the effectiveness of using Interrupt_Priority'Last in the presence non-maskable interrupts. Jorge later mentioned that there are other alternatives, such as forbidding the use of POs during a sliced sequence (this can be checked by the runtime); or adding empty slots at the end of continuation slots, to absorb the potential protected overrun; or to look into implementing a deferred Hold.
- Should delays be forbidden, or checked in TT Tasks?
- Should mode changes be forbidden or checked if they are in the middle of sliced sequences?
- Should this proposal be proposed for standardization?
- Is there no other way to get the benefits of this proposal?
- Are extensions needed to Ravenscar in order to integrate TT scheduling?
- Should a new profile instead be invented for this purpose?
- Should a new Task_Dispatching_Policy be created for this?

- Should this be considered a partition framework or a language element?
- With Ravenscar, can the runtime be trusted or allowed to make certain calls such as Hold/Resume while not allowing the application to make such calls?

Overall the group felt that the proposal provides important capabilities and continued work to improve the solution where possible would be worthwhile.

References

[1] J. Real, S. Sáez, A. Crespo (2018), *Ravenscar Support for Time-Triggered Scheduling*, This Issue.

Session Summary: Deadline Floor Protocol

Andy Wellings (Session Chair)
University of York, UK; email:andy.wellings@york.ac.uk

Jorge Real (Rapporteur)
Universitat Politècnica de València, Spain; email: jorge@disca.upv.es

1 Introduction

This session tackled the definition of a new variant of the EDF task dispatching policy. The current EDF Across Priorities task dispatching policy combines an EDF scheduler with the Stack Resource Protocol (SRP) [1] for controlling the access to protected objects. There is a proposal on the table to replace SRP with the Deadline Floor Protocol (DFP) [2], hence leading to a new task dispatching policy combining EDF and DFP. DFP has all the key properties of SRP, but it is conceptually simpler. The proposal is supported by the ARG, but there is no wording yet for an Ada Issue. The purpose of this session was to agree on the details of an Ada Issue on this topic, to be submitted for consideration towards Ada 202X.

Alan Burns presented the details of the protocol, including a description of DFP and aspects to consider towards embedding the protocol in Ada. Alan also listed some open issues to receive feedback from the workshop towards the writing of a related Ada Issue. The following sections summarise the proposal and reflect the results of the discussion.

2 Deadline Floor Proposal

Alan first summarised DFP. Under DFP, every protected object (PO) has a relative deadline equal to the shortest relative deadline of any task that uses it. This is called the deadline floor of the PO. The idea behind DFP is that the absolute deadline of a task (as used for EDF scheduling) can be temporarily shortened while accessing a PO.

Given a task with absolute deadline d that accesses a resource with deadline floor F at time t, the absolute deadline of the task is (potentially) reduced according to new d ← min(old d, t + F) while holding the PO. The action of the protocol on a single processor with tasks that do not self-suspend results in a single blocking per task, deadlock-free execution, and the protocol works for nested calls on POs as well.

Whilst a task accesses a PO, its deadline is reduced so that no newly released tasks can preempt it and then access the PO. Under a single processor platform, and for tasks that do not self-suspend and have the correct relationship between relative and absolute deadline, this property means that a lock is not strictly required to grant mutual exclusion in the access to POs. However, due to the common use of multiprocessors, a lightweight mutex lock is proposed for DFP.

2.1 Embedding DFP in Ada

Alan then went on with the details of embedding the DFP into the language. With EDF, a task has:

- a last release time, r, when the task was last made ready for execution;
- a relative deadline, D;
- and an absolute deadline, d

When a task is released, its absolute deadline is obtained as d = r + D. And when a task accesses a PO with deadline floor F it must hold that F $\leqslant$ D (or F $\leqslant$ d − r). Alan then distinguished two use cases for the protocol, with regard to general runtime requirements:

Hard real-time systems Here the program sets only the relative deadline, D, either statically or dynamically. Tasks do not self-suspend, except to wait for a new release. The runtime knows the task release time r and computes the absolute deadline d whenever the task is made ready. For this case, on entry to a PO, the runtime is just required to check that F $\leqslant$ D — otherwise, it would be a floor violation.

Soft real-time systems In this case, the program directly manipulates the absolute deadline, d. The task can suspend itself more than once in a single iteration and may not bother with a fixed notion of relative deadline, D — but we still need to have a minimum value for D when computing the floor value for a PO. For this use case, on entry to a PO, the runtime check to make is F $\leqslant$ d − r, hence the runtime needs to retain the release time of the task r (or d − r).

Alan then listed the general requirements to embed DFP in Ada:

- All tasks have a relative deadline (the base deadline) assigned via an aspect or a routine defined in a library package.
- Protected objects must have also a relative deadline (floor) assigned via an aspect.
- Default relative deadline values must be defined for tasks and protected objects (and their types).
- The rules for EDF scheduling must be extended to include a new locking policy: Deadline Floor Locking (or just Floor Locking).
- The rules for EDF scheduling need simplifying to remove the across priorities feature of the current definition, since DFP does not need to support the SRP concept of preemption levels.

- For completeness (and parity with priority ceilings) means of modifying the relative deadline at- tribute of tasks and protected objects should be defined.
- Each PO needs a Deadline Floor attribute that can be set on the creation of a PO by an aspect.

The Relative Deadline aspect already exists, so it can be reused to set the deadline floors. Note that this is identical to the way that the priority aspect is used both for task priority and PO ceiling priority.

2.2 Run-time rules

All these requirements considered, the rules to enforce DFP at run time are:

1. Whenever a task is executing outside a protected action, its active deadline is equal to its base deadline.
2. When a task executes a protected action its active deadline will be reduced to (if it is currently greater than) Now (the current time) plus the deadline floor of the corresponding protected object. Now is obtained via use of the real-time clock.
3. When a task completes a protected action its active deadline returns to the value it had on entry.
4. When a task calls a protected operation, a check is made that (absolute deadline - last release time) of the task is not less than the deadline floor of the object. Program Error is raised if this check fails.
5. When a task is resumed having been suspended on a protected entry call, its active deadline should be no greater than Now + F, where Now is the value of the real-time clock when the task entered the PO, and F is the deadline floor of the PO.

3 Issues considered

Having defined the major aspects of the protocol, Alan then introduced other considerations and raised open issues to be discussed in the AI.

Dynamic deadline floor In the same way that a protected object can contain code to change its own ceiling priority, it can have its minimum deadline floor updated by an assignment to the Relative Deadline attribute under DFP. A task can also change its relative deadline. As with ceiling changes under Ceiling Locking, the change takes effect when the calling task exits the protected object. This was supported by the workshop.

Sporadic tasks With sporadic tasks, the deadline depends on the occurrence of its release event. With clock-triggered releases, there is the Delay Until And Set Deadline operation that can be used to delay and set a new deadline in an atomic operation. But this approach is not feasible for sporadically released tasks as the releaser may be unaware of the releasee's deadline. It was noted that there is the Suspend Until True And Set Deadline operation, useful if the sporadic is released by means of a suspension object, but the issue is when the sporadic is released from an entry.

With EDF, Alan argued, it is probably acceptable to leave the deadline unchanged by the synchronisation operation (delay or entry call) and require the released task itself to modify its deadline, as this code will be executed with a high priority as the current active deadline will be very short (if not in the past). However, this approach may take extra context switches. If the sporadic is released from a PO entry, changing the task's deadline in the entry body works.

But in the hard real-time use case, the reliable approach is to let the runtime automatically calculate the deadlines of sporadic tasks, depending on their release time and relative deadline. The workshop consensus was for a configuration aspect to have the absolute deadline automatically adjusted by the runtime upon calls to delay, delay until, suspension objects or entry calls.

Checking for DFP violations Setting a wrong (too large) floor on a PO can be caught at run time by a check on entry to the PO, pretty much in the way it is done under Ceiling Locking in FIFO Within Priorities. But with EDF and DFP this check requires the last release time of a task to be maintained by the runtime. This is the last time the state of the tasks changed from suspended to runnable.

Last release time The issue here was whether the workshop supported the idea of providing a routine so that a task could obtain its last release time. This would be useful to let the program tasks make decisions about their own timing. The idea was well received as a function call to the runtime.

Combination of Floor Locking and Ceiling Locking Both policies could be used at the same time and hence the existing rules for Ceiling Locking should be slightly modified. Perhaps the policy should be named Ceiling And Floor Locking, or say that Floor it would come implicitly with Ceiling. This aspect was left for Alan to reflect in the AI.

Name of EDF policy Since the concept of preemption levels needs not be supported under DFP, the EDF/DFP dispatching policy should be named EDF Within Priorities (not across).

Deadline inheritance It was agreed that deadlines should be inherited anywhere where priorities are inherited in the language (such as in a rendezvous).

References

[1] T. P. Baker (1991), *Stack-based scheduling of realtime processes*, Real Time Systems, 3(1).

[2] A. Burns, M. Gutiérrez, M. Aldea, and Michael González-Harbour (2015), *A Deadline-Floor Inheritance Protocol for EDF Scheduled Embedded Real-Time Systems with Resource Sharing,* IEEE Transactions on Computers, 64(5):1241–1253.

Session Summary: Language Issues

Tullio Vardanega (Session Chair)
Università di Padova, Italy; email: tullio.vardanega@math.unipd.it
Andy Wellings (Rapporteur)
University of York, UK; email: andy.wellings@york.ac.uk

1 Introduction

The goal of this session was to discuss the Ada 202x proposed changes and focus on those that have not been addressed elsewhere in the Workshop. Tullio Vardanega first gave an overview of the Ada 202x language revision process, which, in ISO speak, is termed an *amendment*. This introduction was followed by a presentation by Brad Moore on the background and motivation for Lock-Free Programming.

2 Ada 202x

Tullio gave an overview of the current status of the Ada 202x amendment proposal. The update to the language is being undertaken under the auspices of ISO/IEC JTC1/SC 22/WG 9 (WG 9 in the sequel). Within WG 9, the Ada Rapporteur Group (ARG) is instituted, which handles comments on the Ada standard (and related standards, such as ASIS) from the general public.

WG 9 instructions to the ARG on the Ada 202x amendment have resulted in Ada Issues (AIs) focusing on:

- Support for parallelism — along the lines proposed by the previous IRTAWs and discussed elsewhere in these proceedings,
- Improved support for contracts — not discussed by the Workshop other than to note that there currently is no direct contract support for real-time or concurrency,
- Improved support for containers — not discussed by the Workshop, and
- Improved support for Unicode and refinements on predefined libraries — not discussed by the Workshop.

As part of the process, however, several other AIs have emerged on a range of other topics, some of which generated from the ARG itself. The three such AIs that are most of interest to the real-time community are:

- AI12-0139-1 — Thread-safe language-defined units, which add variants of the standard libraries that are thread-safe. This addresses the problem that concurrency/parallelism in application software is becoming the normal case, not a specialised one. However, the Ada standard libraries are not required to be thread-safe. Members of the Workshop were pleased to see that this issue was being addressed.
- AI12-0230-1 — Deadline floor protocol, which was generated by the previous IRTAW, see the session summary on the Deadline Floor Protocol in these proceedings.
- AI12-0234-1 — Compare-and-swap for atomic objects, which provides general support for lock-free structures (see Section 3).

The details of the AIs can be found at `http://www.ada-auth.org/ai-files/grab_bag/2020-Amendments.html`. The process is well advanced and WG 9's aim is to agree the scope of the language amendment in June this year (2018). Hence, Tullio stressed that there is no time left for new significant issues to be raised, but small important issues might still be taken on board.

At the time of this workshop, the AIs being considered are at various levels of maturity, which the ARG editor had represented at the cited URL, using the following colour code:

- Black — if ARG approved,
- Green — if wording is proposed (but still may need final approval),
- Orange — if a consensus solution has been described,
- Red – otherwise.

The current status of the AIs of particular interest to the workshop is summarised in Table 1. Note that, although some of these are classified as RED, they were all considered important issues and their respective champions within the ARG were aware of the need to progress the AIs rapidly to at least the Orange status, if they are to make the June cutoff.

Table 1: AIs Status

AI12-0119-1	Parallel blocks and loops	Green
AI12-0242-1	Reduce/Parallel_Reduce attributes	Green
AI12-0251-1	Explicit chunk index for parallel loops	Red
AI12-0251-2	Manual chunking operations	Red
AI12-0262-1	Map-reduce attributes	Red
AI12-0266-1	Parallel container iterators	Green
AI12-0267-1	Data race and blocking prevention	Red
AI12-0139-1	Thread-safe language-defined units	Orange
AI12-0230-1	Deadline floor protocol	Red
AI12-0234-1	Compare-and-swap for atomic objects	Red

3 Lock-Free Programming

Brad indicated that current Ada was lacking in the area of low-level synchronisation primitives such as those needed to support lock-free algorithms, which are important in parallel programming. He presented three approaches by which lock-free programming could be supported by Ada 202x.

- Provide a `Lock_Free` aspect that can be applied to protected objects — a version of this is already implemented by AdaCore.
- Provide an interface to atomic primitives similar to the gcc API for C and C++ — this is an existing API and perhaps more general than needed by Ada.
- For Ravenscar, allow the CPU aspect to be applied to protected objects, to indicate that all users of that PO are on the same core, and therefore could do away with locks because of the priority-ceiling protocol being in force locally — this is a scheduling-based solution which obviously cannot be used to support communication between tasks on different processors.

The workshop considered each of these in term.

Lock-Free Protected Actions

The workshop distinguished between two approaches for achieving lock-free protected actions. Both required certain restrictions to be placed on the Ada application code. In the first approach, the goal was to only support those restrictions that would facilitate an implementation using transactional memory technology or a data replication approach. This approach was unanimously rejected by the Workshop because it complicated timing analysis and memory usage analysis.

The second approach required severe restrictions so that each protected action could be mapped to a single atomic machine operation. Again this was unanimously rejected by the Workshop because it was felt that the approach was too limited.

An interface to the atomic primitives provided by the gcc API

The Workshop briefly discussed the fact that there was a wide range of "lock-free" semantics (wait-free, loop-free etc), and that lock-free approaches tended to focus on lock-free data structures (queue, lists, etc). Therefore, there was unanimous agreement that lock-free algorithms were best supported by providing direct access to the primitive atomic operations. It was also felt that this solution made the code more visible and the corresponding timing properties more apparent.

A CPU aspect for Protected Objects

Although originally proposed for the Ravenscar profile, the Workshop unanimously supported a new aspect specification, valid for all applications, to indicate to the compiler that a particular PO would only be accessed from tasks resident on a single CPU. Since this constraint cannot be asserted at compile time, a run-time check would be required for it during execution.

This issue also led to a general discussion on whether it should be possible to indicate that a PO would only be accessed from a single dispatching domain. This possibility was unanimously rejected by the Workshop because it was felt it didn't provide any more guarantees that the current approach. Also it was noted that, in the session on multiprocessors, the Workshop had decided that all queuing for admissions into a PO should be FIFO so as to facilitate timing analysis.

Brad agreed to progress the associated AI (AI12-0234-1) to reflect the Workshop's view on this topic.

Session Summary: Clock Issues

Kristoffer Nyborg Gregertsen (Session Chair)
SINTEF Digital, Trondheim, Norway; email: kristoffer.gregertsen@sintef.no

Luis Miguel Pinho (Rapporteur)
CISTER/ISEP, Portual; email: lmp@isep.ipp.pt

1 Introduction

The session was based on a position paper by Kristoffer Nyborg Gregertsen [1], on clock support in Ada. Kristoffer started by providing a brief overview of time and clocks, presenting issues such as resolution, precision, accuracy and drift. The presentation continued with some examples, which led to motivate the need to support high-precision distributed time:

- Time synchronized events in distributed control systems;
- Sensors with high-precision timestamps for monitoring physical processes;
- Applications in robotics, process automation, smart grid applications, etc.

Kristoffer then summarized the support for calendar and clock in Ada

- The Calendar package, implemented by the system clock;
- Then real-time clocks, which are required to be monotonic with documented drift;
- And execution-time clocks and timers, for CPU time, tasks and interrupts.

At this point, Kristoffer raised the issue that there is no support to dynamic clock rates or different rates on CPU cores. Michael González Harbour noted that they had already faced this problem and had to disable dynamic clock rates, so a solution for this issue would be important. Michael also noted that Ada allows implementations to add additional time types as an extension that can be used in delay statements.

2 Issues Discussed

Kristoffer presented a few difficulties with clocks and timers in Ada, for which he would like to propose changes:

- There is no standard way of acquiring high-precision timestamps;
- It is not possible to relate the real-time monotonic clock to UTC/TAI;
- The Calendar package is not synchronized with UTC;
- Calendar is not allowed in Ravenscar systems;
- There are no explicit clock types with common interface, only clock functions for different time types;
- Timer are defined as tagged types, but have no common interface and have subtle differences (e.g. timing event, timer and group budgets).

Michael noted that implementation can actually implement a real-time clock that synchronizes to TAI and a calendar that synchronizes with UTC, although not forced to. Then, Andy Wellings put forward that applications can also do this synchronization with a high-priority task doing time synchronization. Nevertheless, Kristoffer considers that there should be a standard way, even to know the discrepancy between clocks.

Concerning Ravenscar, Calendar is not allowed, which means that in Ravenscar it is not possible to reason on wall time. Kristoffer provided an example of a smart grid system, where it could be required to open a switch at a specific time instance in the day.

There was also some discussion on the fact that there are no explicit clock types and the interfaces of timers are different, which makes it difficult to provide a common hierarchy. Time could be a root type, where other time types derive (they are private). But the finer details still need to be looked at (e.g. Calendar Duration and real-time Time_Span are different types for referring to a time interval).

Afterwards, the workshop discussed the issue of execution-time timers for interrupts, a theme recurrent from previous workshops. The existing possibilities were reviewed, but no further solutions were considered. There was also some discussion on execution time and parallelism: having the possibility to know post-fact the execution time of parallel computation could be interesting. Not so much, the ability to fire event handlers on execution-time overruns.

At this point, Kristoffer proposed to have coherent clock and timers in Ada, where clocks could be defined as explicit tagged types with a shared interface. This would make it easier to understand rather than having subtle differences, other clocks could be defined with special properties such as UTC or TAI and tailored clocks for particular systems or applications. Kristoffer also compared with the support to clocks and timers in C++11 and 20 and RTSJ 2.0, noting the advances provided in these languages.

In the discussion it was also raised that it would be important at least to be able to detect loss of synchronization with a time source (e.g. NTP or GPS), either with an interface where this could be queried or the possibility to raise an exception if the program tried to use a clock which has lost synchronization.

At the end of the session it was generally agreed that it would be important for Ada to have more advanced support for dealing with time, as the premier real-time language. An argument was made that if Ada lags behind in important things where other languages are advancing it will be difficult to attract programmers.

Nevertheless, due to the time-lag in standardization, having a non-standard peer-reviewed library would be interesting as an incubator for new ideas before going to the Ada Rapporteur Group (ARG). It was generally agreed that the proposed extensions should be provided first under the "XAda" package hierarchy that the workshop decided to endorse in the Time Triggered Scheduling session [2].

References

[1] K. N. Gregertsen (2018), *Position paper: Clock support in Ada*, This Issue.

[2] J. Real, B. Moore, Session *Summary: Time Triggered Scheduling in Ravenscar*, This Issue.

Combining the tasklet model with OpenMP

Luis Miguel Pinho

CISTER/ISEP, Portugal; email:lmp@isep.ipp.pt

Eduardo Quiñones, Sara Royuela

BSC, Spain; email:{eduardo.quinones,sara.royuela}@bsc.es

Abstract

Previous workshops have discussed a proposal to augment Ada with fine-grained parallelism, based on the notion of tasklets, a lightweight parallel entity. Recent works have shown the convergence of this model with the OpenMP tasking model and have proposed their coexistence. In this paper we provide a status of the existent works, and describe how these models could be combined.

1 The Tasklet Model

The existent proposal to extend Ada with a fine-grained parallelism model is based on the notion of tasklets [1], lightweight computation units, which would allow the specification of potential parallelism, not fully controlled by the programmer, but under the control of the compiler and the runtime. In that regard, the tasklet model follows the same principle of other parallel tasking models used in the general purpose and high-performance domains, in which the programmer uses special syntax to indicate where parallelism opportunities occur in the code, whilst the compiler and runtime co-operate to provide parallel execution, when possible.

In the tasklet model, each Ada task is a graph of multiple tasklets using a fully-strict fork-join model [2] (Figure 1). Tasklets can be spawned by other tasklets (fork), and need to synchronize with the spawning tasklet (join). Tasklets as defined are orthogonal to Ada tasks and execute within the semantic context of the task from which they have been spawned, whilst inheriting the properties of the task such as identification, priority and deadline. The concept

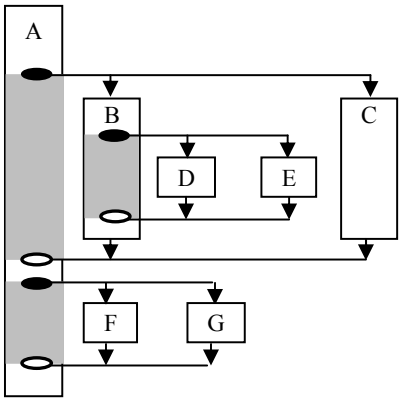


Figure 1. Task DAG example following a fork-join model [4]

is that the model allows a complete graph of potential parallel execution to be extracted during the compilation phase.

Together with the Global aspects proposed in [3], it is thus possible to manage the mapping of tasklets and data allocation, as well as prevent unprotected parallel access to shared variables. Although not a topic addressed in [3], the work considers that issues such as data allocation and contention for hardware resources are key challenges for parallel systems, and therefore compilers and tools must have more information on the dependencies between the parallel computations, as well as data, to be able to generate more efficient programs.

The tasklet execution model (Figure 2) is based on the notion of abstract executors [4], which carry the actual execution of Ada tasks in the platform, under different

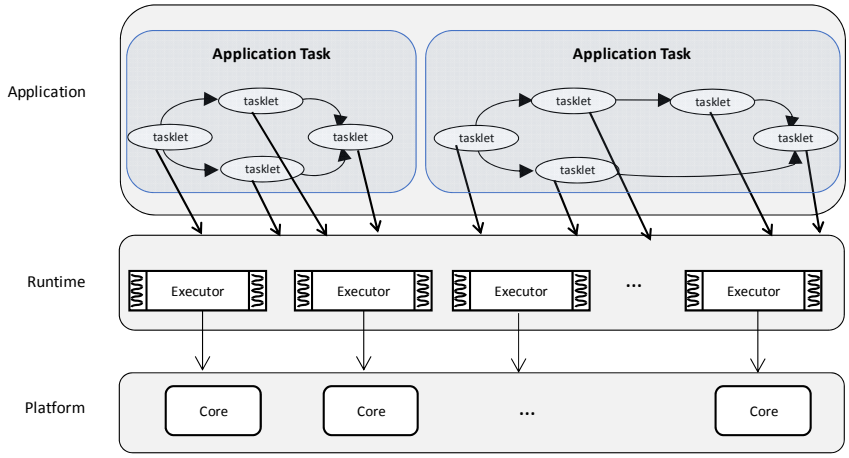


Figure 2. Tasklet execution model [4]

progress guarantees that the compiler and runtime need to provide to the parallel execution. The model also specifies that calls by different tasklets of the same task into the same protected object are treated as different calls resulting in distinct protected actions, enabling to synchronize tasklets with protected operations [4].

The tasklet model is being currently considered for standardization, with a set of Ada Issues (AI) being discussed within the Ada Rapporteur Group (ARG). The tasklet model itself, as well as the Ada constructs for parallel execution, are specified in AI12-0119-1 [5]. A relevant difference to the model proposed in [4] is that tasklets are not allowed to perform potentially blocking operations.

2 Combining the OpenMP Tasking Model and the Tasklet Model

Recently, a work [6] has analysed the similarities of the tasklet model with the OpenMP tasking model [7]. In this work, the term task in OpenMP is not related to Ada tasks but to tasklets, as OpenMP tasks are lightweight parts of the code that can be executed in parallel by worker threads. The tasking model appears in OpenMP 3.0 from the need of efficiently and easily implementing certain types of parallelism: unbounded loops, recursion, unstructured parallelism, etc., which clearly complement the structured parallelism approach of the tasklet model.

The OpenMP tasking model follows the same principle of the tasklet model, where the compiler and the runtime system are the ones responsible for generating and executing the OpenMP tasks, based on specific OpenMP constraints (e.g. control-flow or data-flow dependencies) and thread availability. As shown in [8], the forms of parallelism defined by OpenMP are compatible with those proposed for Ada tasklets. Furthermore, the relaxed fork-join model defined in OpenMP, in front of the strict model defined for Ada tasklets, allows for a more flexible execution, as spawn and distribution operations are not done at the same point of the execution (Figure 3). That is, in Ada, the same parallel statement, i.e., parallel do and parallel loop, is in charge of spawning and distributing the computation among the executors (Figure 3a). OpenMP, instead, defines the parallel construct to spawn work, and several constructs (e.g., for, task, taskloop) to distribute this work among threads (Figure 3b shows how two parallel loops can run in parallel with each other).

Interestingly, the work in [8] allowed verifying that the execution model and the memory model of both OpenMP and Ada tasklets are compatible, hence enabling the OpenMP runtime to be used to implement the Ada tasklet model, as well as to introduce additional functionalities, in particular for unstructured fine-grained parallelism (with fine-grained synchronization mechanisms such as task dependences) and heterogeneity (with the target construct for offloading synchronous and asynchronous tasks). Furthermore, [8] proves that the use of OpenMP functionalities not provided in the tasklet model allow enhancing the performance possibilities of non-embarrassingly parallel applications.

More recently this work has been extended [9] to show that compiler techniques can identify race conditions that can potentially appear in Ada programs parallelized with both OpenMP and/or Ada tasks. Moreover, this works can be used for checking the correctness of the OpenMP directives regarding dependence clauses and data-sharing attributes. Overall, this work enables to ensure safety in the presence of parallel computation.

To sum up, it has been proven that it is possible to provide Ada with two separate, but compatible, models:

- The tasklet model, to be used for homogenous regular parallelism, and that is included in the language core, as a key mechanism for parallelism.
- OpenMP, as an external model to the language, which can be used both as a runtime to support the tasklet model, as well as a complementary parallel programming model, providing more complex and flexible parallelism, and the support for heterogeneity.

For the latter, the integration of OpenMP and Ada would not be done in the Ada standard, but preferably as an additional language in the OpenMP specification.

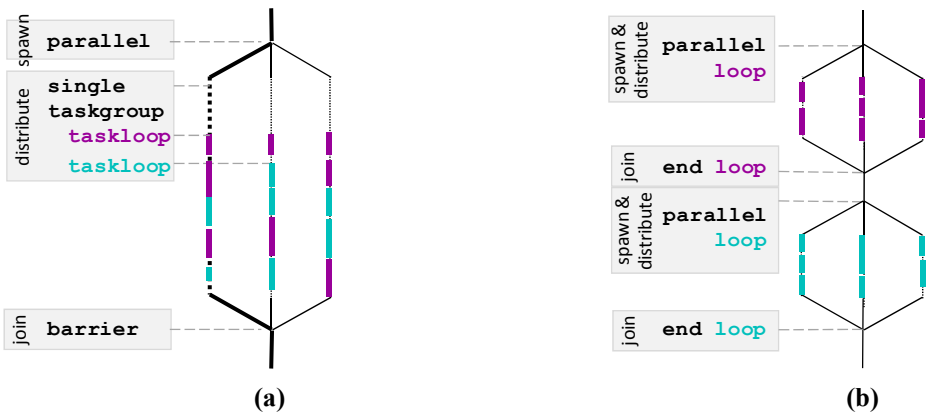


Figure 3. Tasklet (a) and OpenMP(b) forms of parallelism [8]

Nevertheless, the integration of the Ada runtime with the OpenMP library requires understanding and controlling the interaction between Ada tasks and OpenMP threads, which raises several challenges [5]:

- Scheduling decisions. For instance, when an Ada task executing parallel code within the OpenMP runtime is preempted, the parallel computation should be also preempted, but OpenMP threads do not have the concept of priority (this concept is attributed to the tasks, in OpenMP). A mechanism needs to be provided that allows for communication between the two worlds.
- Protected objects access. An Ada tasklet, being executed by the OpenMP runtime, can access Protected Objects (POs), which requires the OpenMP library to be aware of the use of POs.
- Access to task attributes. Calls to Ada Task attributes need to use the context of the Ada Task, but can be made from within OpenMP threads.

Another aspect analyzed in [8] is the possibility of blocking and preempting Ada tasklets and hence OpenMP tasks. That work shows how OpenMP enables to mimic the use of blocking operations within tasklets by means of introducing task scheduling points (moments at which a thread can stop executing a specific task and start executing a different one). Nevertheless, the current proposal for the Ada standard [5] does not allow the use of blocking operations within parallel blocks (thus disabling preemption), so this is no longer an issue.

It is worth mentioning that there are ongoing discussions within the OpenMP community regarding the use of OpenMP in safety critical environments [10], and in particular the tasking model, which could be a good application domain for this integration.

Acknowledgements

This work was supported by National Funds through FCT (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (CEC/04234), the European Union's Horizon 2020 research and innovation programme under the grant agreement No 780622, and the Spanish Ministry of Science and Innovation under contract TIN2015-65316-P.

References

[1] 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.

[2] 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.

[3] 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 (HILT '14). ACM, New York.

[4] 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.

[5] AI12-0119-1/08, Parallel operations, http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai12s/ai12-0119-1.txt, last accessed April 2018

[6] S. Royuela, C. Martorell, X, E. Quiñones, L. M. Pinho (2017), *OpenMP tasking model for Ada: safety and correctness*, 22nd International Conference on Reliable Software Technologies (Ada-Europe 2017). 12 to 16, Jun, 2017, pp 184-200. Vienna, Austria. DOI: 10.1007/978-3-319-60588-3_12.

[7] OpenMP Architecture Review Board (2018), *OpenMP Application Program Interface*, Version 4.5, November 2015, available at http://www.openmp.org/mp-documents/openmp-4.5.pdf, last accessed January 2018.

[8] S. Royuela, L. M. Pinho, E. Quinones (2018), *Converging Safety and High-performance Domains: Integrating OpenMP into Ada*, In the Design, Automation, and Test in Europe conference (DATE). Dresden (Germany), March 19-23.

[9] S. Royuela, X. Martorell, E. Quiñones, L. M. Pinho (2018), *Safe Parallelism: Compiler Analysis Techniques for Ada and OpenMP*, 23rd International Conference on Reliable Software Technologies (Ada-Europe 2018),Lisbon, Portugal.

[10] S. Royuela, A. Duran, M. A. Serrano, E. Quiñones, X. Martorell (2017), *A functional safety OpenMP for critical real-time embedded systems*, In the 13th International Workshop on OpenMP (IWOMP), New York (USA).

Synchronous Signals: An Abstraction for Interleaving Sequential and Parallel Code

Brad Moore

General Dynamics, Canada; email:brad.moore@gdcanada.com

Abstract

In Ada 2012, the language expanded its support for concurrency with the addition of the Synchronous Barriers library package to the Real-Time Systems annex[1]. This package provides a mechanism to synchronize a group of tasks after the number of blocked tasks reaches a specified count value. One use for this feature is to interleave sequential processing with concurrent or parallel processing. For this usage, two synchronous barrier objects can be utilized where one barrier manages the transition from parallel to sequential code, and the other barrier manages the transition back from sequential to parallel code. In general, performance can be improved by minimizing the amount of synchronization in an application. The more that threads of execution can proceed independently without interference with other threads, the more likely that the available CPUs can focus on completing the independent tasks rather than spending time synchronizing with the other threads. A Synchronous Signal is a synchronization primitive that provides a similar abstraction as a Synchronous Barrier, except it can reduce the amount of synchronization needed by a factor of two. In addition, only one object is needed, to manage both transitions instead of two. In this paper, the abstraction is explored and an interface to use the abstraction is presented. Two forms of the abstraction are considered; a blocking form and a non-blocking form, and the performance measurements are reported and compared against Synchronous Barriers usage. Finally, these examples are also compared and considered for use in a Ravenscar environment.

1 Introduction

There exist certain algorithms that involve parallel or concurrent processing, but where sequential processing is also needed during certain points of the processing. That is, a single thread of execution is needed perhaps to provide some summarization or preparation for the next round of concurrency.

In this scenario, typically synchronization is needed to ensure that the sequential processing does not proceed until the preceding concurrent processing has completed. Similarly, the concurrency should not be restarted until the sequential processing is complete.

Ada 2012 provides a standard library package, Ada.Synchronous_Barriers, that can be used to provide the needed synchronization. This can be accomplished by declaring two barrier objects, one that manages the transition from concurrent to sequential processing, and the other that manages the transition from sequential back to concurrent processing.

For example, consider a basic weather prediction simulation system where the current time is advanced in small increments, and the state for that time is then computed.

```
procedure Weather_Prediction is
   ...
   Barr1, Barr2 : Synchronous_Barrier
    (Release_Threshold =>
        Number_Of_Weather_Stations);

   task type Weather_Station;

   task body Weather_Station is
     Notified : Boolean;
   begin
     Simulation_Loop : loop
       Update_Local_Weather;
       -- Computationally expensive done in parallel

       -- Transition to sequential
       Ada.Synchronous_Barriers.Wait_For_Release
       (The_Barrier => Barr1, Notified    => Notified);
       if Notified then
         Current_Time := Current_Time +
       Simulation_Time_Increment;
         Generate_Weather_Map (Current_Time);

         if Current_Time >= End_Time then
           Done := True;
         end if;
       end if;

       -- Transition back to parallel processing
       Ada.Synchronous_Barriers.Wait_For_Release
       (The_Barrier => Barr2, Notified    => Notified);
       exit Simulation_Loop when Done;

     end loop Simulation_Loop;
   end Weather_Station;
begin -- Weather_Prediction
   declare
```

```
      Weather_Stations :
        array (1 .. Number_Of_Weather_Stations) of
          Weather_Station;
    begin
      null;
    end;
    ...
  end Weather_Prediction;
```

In this example, we see that there are N * 2 task synchronizations in each iteration of the Simulation_Loop, since all N tasks must synchronize before entering the sequential processing and again after leaving the sequential processing.

It is desirable to reduce the number of synchronizations if possible to make this work.

This paper proposes a Synchronous_Signals abstraction that can reduce the number of synchronizations to N, thereby reducing the synchronization count by 50%. In addition only one synchronization object is needed to manage both transitions, instead of two.

The paper describes a few algorithms where this pattern can be applied to improve performance on multicore architecture, then evaluates the performance against the existing package Ada.Synchronized_Barriers. It then considers whether non-blocking forms can further improve performance, and finally assesses whether the construct could be applied to Ravenscar compliant applications.

An open source reference implementation of these interfaces can be found in Paraffin[8], a suite of generics adapted to the Ada 2012 standard. Note that "Signal" may not be the most appropriate name for this abstraction, as it can apply to a broader category of synchronization utilities, such as transient signals [3] in which a signaller can signal any number of waiting signalees.

2 Synchronous Signals Interface

A Synchronous_Signal object is similar to a Synchronous_Barrier in that its purpose is to synchronously release a group of tasks after the number of blocked tasks reaches a specified count value, except that the release occurs after sequential processing has been applied instead of before.

Unlike Synchronous_Barriers, only a single Synchronous_Signals object is needed to manage two transitions, where the first transition is from parallel to sequential execution, and the second transition is from sequential back to parallel execution. With Synchronous_Barriers, two objects are needed, one for each transition.

The package providing this interface is as follows;

```
package Synchronous_Signals is

   subtype Synchronous_Task_Count is Natural;

   type Synchronous_Signal
     (Task_Count : Signal_Limit) is limited private;

   procedure Send_And_Wait_For_Reset
     (The_Signal : in out Synchronous_Signal;
      Id         : Worker_Id);

   procedure Wait_For_Release
     (The_Signal : in out Synchronous_Signal);

   procedure Reset
     (The_Signal : in out Synchronous_Signal);

private
   ...
end Synchronous_Signals;
```

The Task_Count discriminant corresponds to the Release_Threshold discriminant of the Synchronous_Barriers package. It identifies the number of tasks that are interacting with the Synchronous_Signal object.

Each task associated with the Synchronous_Signal object is expected to be assigned a unique index number in the range of 1 .. Task_Count. All tasks with an index number less than Task_Count are expected to be the "signallers", and the task with unique index value equal to Task_Count is expected to be the "signalee". When transitioning from parallel execution to sequential execution, the "signalee" is expected to call Wait_For_Release, which blocks until all the "signallers" have each issued the call to Send_And_Wait_For_Reset. The "Signallee" is then responsible for performing any sequential processing that needs to be completed, then signals the "signallers" by calling Reset, which releases the "Signallers" to resume parallel execution, and since Reset is not a blocking call, the "Signalee" also continues parallel execution with the other tasks without blocking or acquiring a lock.

The act of signalling other tasks is a lightweight operation that does not involve synchronisation on the part of the task sending the signal. Sending a signal essentially involves just setting a Boolean flag associated with the signal sender that can be read by the other tasks.

On the first transition from parallel to sequential, only one task, the "signalee", blocks while waiting for all the "signalers" to report in. On the second transition from sequential to parallel, Task_Count – 1 tasks block as all the "signallers" block waiting for the "signalee" to call Reset.

For both transitions, the total number of synchronizations is N (Task_Count), whereas for Synchronous_Barriers there are 2N (Release_Threshold * 2) synchronizations since all tasks need to synchronize on both transition points.

3 Applications of Use

The Synchronous_Signals abstraction is useful in situations typically where there a parallel loop requires sequential processing inside the loop, or when there is an

outer loop that cannot be executed in parallel, but contains an inner nested loop that needs to be executed in parallel.

This second case is a bit more complex, as it typically involves enclosing the loops with a parallel loop, and using synchronous signals to transition between sequential and parallel processing, instead of making the innermost loop the parallel loop.

The reason for doing this is because there is overhead associated with starting up the parallelism, and for the parallelism to be worthwhile, the processing of the loop needs to be significantly more time consuming than the amount of processing associated with the startup overhead. By having the parallel loop nested inside a sequential loop, the parallelism overhead is multiplied by the number of iterations of the sequential loop. By enclosing two loops with a parallel loop, this reduces the number of occurrences of parallelism overhead to just one, while using synchronous signals to transition between sequential and parallel processing inside the inner loops is generally lighter weight than the startup parallelism overhead.

The Weather simulation example above is one such example of this situation. A sequential version of that application might have started with an outer sequential loop that updates the current time, then calls an inner loop to update the current weather at all weather stations based on the current map, and the time increment into the future. By rearranging the source so that the forking of parallel processing occurs outside the outer loop, and the sequential processing for the next iteration occurs at a lower level, the overhead of starting up the workers for all the weather stations only occurs once.

To see how the previous example looks after replacing the use of Synchronous_Barriers with Synchronous_Signals, we see;

```
procedure Weather_Prediction is
   ...
   Signal : Synchronous_Signals.Synchronous_Signal
     (Task_Count => Number_Of_Weather_Stations);

   task type Weather_Station (Worker_Index :
        Worker_Id := Worker_Id'First);

   task body Weather_Station is
   begin
     Simulation_Loop : loop
       Update_Local_Weather;
       -- Computationally expensive done in parallel

       if Worker_Index < Number_Of_Weather_Stations
then

Synchronous_Signals.Send_And_Wait_For_Reset
            (The_Signal => Signal,
             Id => Worker_Index);
       else
         -- Sequential Section
            Parallel.Synchronous_Signals.
                  Wait_For_Release (Signal);
          Current_Time := Current_Time +
                  Simulation_Time_Increment;
          Generate_Weather_Map (Current_Time);

          if Current_Time >= End_Time then
            Done := True;
          end if;

          Synchronous_Signals.Reset (Signal);
        end if;
        exit Simulation_Loop when Done;

     end loop Simulation_Loop;
   end Weather_Station;
begin -- Weather_Prediction
   declare
     Weather_Stations :
       array (1 .. Number_Of_Weather_Stations)
        of Weather_Station;
   begin
     null;
   end;
   ...
end Weather_Prediction;
```

This pattern is likely fairly common for simulation work. Another example is the n-body simulations in physics and astronomy where particle locations are updated based on physical forces such as gravity.

The outer loop for such simulations involves stepping the current time into the future using small time increments. This typically cannot be a parallel loop, since the state for the next point in time for all the particles depends on the state from the previous time increment. The process of updating the state of all the particles however that needs to happen for each time increment can be parallelized since the new location of the particle is based entirely on the state of the particles from the previous time increment.

Another example of this pattern is applying parallelism to matrix solving using Gaussian Elimination. In that algorithm, the outer loop involves iterating through the columns of the matrix looking for so called “pivots” but that loop must be a sequential loop since one cannot proceed to the next column until the pivot has been determined and the row operations applied to progress towards row-echelon form. Similarly, the use of Synchronous Signals can be applied so that the outer loop is divided into row chunks, and the iteration through columns happens as an internal loop during sequential processing phase of the Signal.

Yet another example for use is prefix scan, or cumulative sum of an array of values. However, there are many ways that parallelism could be applied to solve that problem, some of which are perhaps simpler to understand and do not involve the use of barriers or signals, so that it is only mentioned here in passing.

4 Performance Measurement

It is important to provide measurements that validate the assertions that using less synchronization via Synchronous Signals should result in better performance than using Synchronous Barriers.

4.1 Matrix Solving using Gaussian Elimination

For this purpose, we will examine the times to solve matrices using Gaussian Elimination, as well as Nbody simulation, where the only differences in approach is the implementation of thread synchronization. Firstly, consider the matrix solving problem. We show the time taken to execute the algorithm using sequential processing, then we show the time using a blocking Signals implementation, followed by a blocking Barriers application, and then followed using the GNAT implementation of Synchronous_Barriers. Finally we show the times using a spinlock Barriers implementation followed with a spinlock Signals implementation, where the synchronization involves a busy wait instead of task suspension.

```
********** Parallel Framework Test **********
  Physical Processors= 4
  Workers = 0
  Effective_Workers = 4

(- Matrix Tests -)

Sequential Gauss Jordan
wall time= 0.839479000 seconds, N= 1000,
processors=1, maxerr:= 1.26004E-04

Work Sharing Signals Gauss Jordan
wall time= 0.455542000 seconds, N= 1000,
processors= 4, workers= 4, maxerr:= 1.35404E-03

Work Sharing Barriers Gauss Jordan
wall time= 0.431885000 seconds, N= 1000,
processors= 4, workers= 4, maxerr:= 2.14815E-04

Work Sharing Ada Barriers Gauss Jordan
wall time= 0.395343000 seconds, N= 1000,
processors= 4, workers= 4, maxerr:= 7.20087E-04

Spinlock Barriers Work Sharing Gauss Jordan
wall time= 0.344413000 seconds, N= 1000,
processors= 4, workers= 4, maxerr:= 1.11427E-04

Spinlock Signals Work Sharing Gauss Jordan
wall time= 0.337744000 seconds, N= 1000,
processors= 4, workers= 4, maxerr:= 4.19378E-04
```

This shows a typical execution. After numerous repetitions of execution, the Blocking Synchronous signals implementation was slightly behind the Blocking Synchronous_Barriers implementation more often than not. Both implementations use a protected object to manage the synchronization. The GNAT Synchronous_Signals implementation was consistently better than both of the blocking implementations, however both spinlock implementations were consistently better than the GNAT implementation. The Spinlock_Signals implementation consistently had the best performance.

This suggests that for this problem at least, the most significant performance gains are based on avoiding protected operations entirely, but also that there appears to be a more modest performance gain through the use of the Synchronous_Signals over the Synchronous_Barriers, but only if protected operations are avoided.

4.2 Discrete N-Body Event Simulation

For this problem, a parallel implementation of Nbody simulation involving 20 particles projecting their locations in three dimensional space forward in time for 100_000 iterations. The following execution times were collected, dividing the work evenly between 4 worker tasks.

The first attempt tried applying 16 worker tasks to 4 CPU cores. This took extremely long for the spinlock implementations. For example, while the blocking implementations were executing around the 40 seconds mark, the spinlock signal implementation had to be dropped from 100_000 iterations to 2_000, and even then the execution time was 62.57 seconds.

It was then realized that the problem was due to having 16 workers, but only 4 cores. Having more workers than cores means that there are multiple workers on the same core, and when a worker completes its task, its busy wait is using CPU that would be better served by the remaining workers who are doing real work. Reducing the number of workers to match the number of cores gave a remarkable speedup. With 100_000 iterations, the execution time is 5.08 seconds.

```
Sequential (1 core): 18.37 seconds
Ada.Synchronous_Barriers: 8.16 seconds
Paraffin.Synchronous_Barriers: 9.39 seconds
Paraffin.Synchronous_Blocking_Signals: 9.22 seconds
Paraffin.Synchronous_Spinlock_Barriers: 4.88 seconds
Paraffin.Synchronous_Spinlock_Signals: 5.08 seconds.
```

Here the results are mostly comparable to the matrix results. Using a spinlock implementation provided significantly better results than the blocking implementations. The use of barriers vs signal was not much of a distinguishing factor, however. The GNAT implementation of Synchronous_Barriers appears to be marginally better than the blocking protected object implementations but had significantly higher execution times than the spinlock versions. It might be worth considering if it would be worthwhile to add additional synchronization mechanisms to the standard, or to at least provide some configuration pragmas or aspects to tune the synchronizations for optimal performance.

5 Ravenscar Consideration

The Ravenscar profile in Ada applies a set of tasking restrictions to the Ada runtime to provide a simpler tasking model[10] that better facilitates analysis for properties such as schedulability. Parallelism libraries can be written for the Ravenscar runtime, and implementations do exist, such as Paraffin. One of the restrictions of the Ravenscar Profile is that there not allowed to be any dependence or usage of the package Ada.Synchronous_Barriers[2]. For the testing above, I used the Paraffin implementation for the Synchronous_Barrier package, so in theory one could use such a package to get around the Ravenscar restriction. However, that being said, other Ravenscar restrictions prevented the use of the Paraffin blocking Synchronous_Signals and Synchronout_Barriers libraries with Ravenscar because they utilize protected objects that are not declared at library level. However, the Ravenscar profile does not restrict the use of other forms of synchronization such as the use of atomic or volatile variables. The spinlock versions of the barrier and signal libraries thus can be applied to Ravenscar applications, while remaining compliant with the profile. Furthermore, a spinlock approach might be desired in Ravenscar anyway, if the parallelism is to be considered processing that occurs as part of the parent task. By spinning, the parent task could spin while waiting for the results and other cores associated with subtasks to the same task at the same priority could be used to carry the parallel tasks of the algorithm. Since all such tasks wouldn't block during this synchronization, this might facilitate analysis, since the whole processing does not involve blocking, allowing for analysis such as commonly used for periodic tasking to be applied. The following shows test results for a Ravenscar compliant execution using a spinlock Synchronous_Barrier implementation and a spinlock Synchronous_Signal implementation.

```
************ Parallel Framework Test *************
  Physical Processors= 4

(- Matrix Tests -)

Sequential Gauss Jordan, Elapsed= 0.90, maxerr:=
1.26004E-04
Barriers Work Sharing Gauss Jordan, Elapsed= 0.67,
maxerr:= 1.35404E-03
Signals Wait Free Work Sharing Gauss Jordan,
Elapsed= 0.67, maxerr:= 2.14815E-04
```

In this case, both synchronization implementations were approximately equivalent in terms of processing time.

6 A Possible Implementation

The following shows a potential implementation for the spinlock version of the Synchronous_Signals package. As can be seen, a simple implementation is possible.

```
package Synchronous_Spinlock_Signals is

  subtype Synchronous_Task_Count is Positive;

  type Synchronous_Signal
   (Task_Count : Synchronous_Task_Count) is limited
private;

  procedure Wait_For_Release
   (The_Signal : in out Synchronous_Signal);

  procedure Reset
   (The_Signal : in out Synchronous_Signal);

  procedure Send_And_Wait_For_Reset
   (The_Signal : in out Synchronous_Signal;
    Id : Worker_Id);

private

  type Worker_Progress_Array is array
      (Synchronous_Task_Count range <>)
      of Boolean;
  pragma Volatile_Components
      (Worker_Progress_Array);

  type Synchronous_Signal (Task_Count :
      Synchronous_Task_Count) is limited
    record
      Gate : Worker_Progress_Array (1 .. Task_Count)
       := (others => False);
    end record;

end Synchronous_Spinlock_Signals;

package body Synchronous_Spinlock_Signals is
  procedure Reset (The_Signal : in out
      Synchronous_Signal) is
  begin
    The_Signal.Gate := (others => False);
  end Reset;

  procedure Wait_For_Release (The_Signal : in out
      Synchronous_Signal) is
  begin
    Spinloop : loop
      <<Try_Again>>
      for I in 1 .. The_Signal.Gate'Last - 1 loop
        if not The_Signal.Gate (I) then
          goto Try_Again;
        end if;
      end loop;

      exit Spinloop;
    end loop Spinloop;
  end Wait_For_Release;

  procedure Send_And_Wait_For_Reset (
      The_Signal : in out Synchronous_Signal;
      Id : Worker_Id) is
```

```
    begin
      The_Signal.Gate (Positive (Id)) := True;

      while The_Signal.Gate (Positive (Id)) loop
        null;
      end loop;
    end Send_And_Wait_For_Reset;

  end Synchronous_Spinlock_Signals;
```

7 Conclusions

In this paper, a Synchronous_Signals abstraction was presented and compared against the existing interfaces provided the existing language defined Ada.Synchronous_Barriers package. Testing of the abstractions show that there is not a significant difference in performance between Synchronous_Barrier implementations and Synchronous_Signal implementations, if both are implemented with the same blocking strategy. However, the paper does show that non-blocking busy-wait versions of these abstractions performs significantly better than the blocking versions, for the algorithms that were examined. There may be value in providing a standard mechanism to specify whether blocking or busy wait implementations should be applied to a problem. Further, using a busy wait strategy might better facilitate timing analysis for schedulability concerns.

One important consideration for spinlock implementations is that it is much more important that the number of worker tasks not exceed the number of available CPU, to avoid busy waits of idle tasks taking CPU away from workers that are performing actual work. It may be useful to provide a standard mechanism to query to the system to determine how many of the available CPU are idle, since CPU's that are busy likely should not be interfered with workers that perform busy waits. The Linux OS for instance allows one to check the current load on the CPU's via the */proc/loadavg* filesystem.

Lastly, while there may be valid reasons for disallowing the blocking forms of Synchronous_Barriers for use with the Ravenscar Profile, it may be worth considering whether busy wait implementations of the abstraction are better suited for use with the Ravenscar Profile, and whether the standard Synchronous_Barrier package should be allowed if it could be configured to correspond to such an implementation. It may be worth considering whether a Synchronous Signal package should be added to the set of language defined packages to increase the synchronisation facilities at the disposal of the programmer. On the other hand, it should be weighed to assess how often such a feature might be used, as well as whether it is trivial enough to let 3rd party library package writers provide such facilities. It may be that the compiler could provide extra safety guarantees that a compiler writes would not be available to 3rd party library writers. To justify that addition, likely it would mean finding an example where a Synchronous_Signals implementation has significantly better performance than a Synchronous_Barrier implementation.

Another consideration is that the Synchronous_Signal abstraction requires assigning a unique index value to each of the participating worker tasks. While this can be managed by the programmer, it might make sense to provide a standard way to obtain such a worker id value. The existing Synchronous_Barriers abstraction does not have this need.

Finally, it might be worth considering whether finer control such as whether to use spinlock synchronization or blocking synchronization could be specified via a configuration pragma or possibly an aspect, that could be applied to uses of Synchronised_Barrier objects.

References

[1] ISO/IEC (2012), *Ada Reference Manual*, ISO/IEC 8652:2012(E).

[2] A. Burns, A.J. Wellings (2016), *Synchronous Task Control and Synchronous Barriers*, ACM SIGAda Letters, Volume 36 Issue 1.

[3] A. Burns, A.J. Wellings, *Concurrent and Real-Time Programming in Ada*.

[4] B. Moore (2010), *Parallelism Generics for Ada 2005 and Beyond.*, SIGAda'10 Proceedings of the ACM SIGAda annual conference.

[5] B. Moore, *Paraffin source libraries*. http://sourceforge.net/projects/paraffin/?source=directory

[6] F. Chouteau, J. F. Ruiz (2011), *Design and Implementation of a Ravenscar Extension for Multiprocessors,* In: Reliable Software Technologies – Ada-Europe 2011, pp 31-45.

On Protocols for Accessing Protected Objects on Multiprocessors*

Jorge Garrido, Juan Zamorano, Juan A. de la Puente

Sistemas de Tiempo Real e Ingeniería de Servicios Telemáticos (STRAST), Information Processing and Telecommunications Centre (IPTC), Universidad Politécnica de Madrid (UPM); email: jgarrido@dit.upm.es, jzamora@datsi.fi.upm.es, jpuente@dit.upm.es

Abstract

This paper discusses different issues related to protected object access in Ada. In particular, the influence of the different approaches on the implementation and analysis of real-time systems is addressed. Based on results from previous and on-going work, some issues are discussed and some proposals are made.

1 Introduction

The topic of scheduling real-time tasks on multiprocessors has received a great deal of attention [7]. An important aspect in this context is communication and synchronization with shared objects, and the use of suitable access protocols for bounding the duration of blocking introduced by priority inversion. On uniprocessor systems, the Priority Ceiling Protocol (PCP) [14] and the Stack Resource Policy (SRP) [3] are generally accepted as the most appropriate access protocols, as they exhibit some interesting properties:

- Deadlock free execution.
- Only tasks of higher priority can preempt running tasks and only if they do not share any locked resource.
- Tasks of lower priority can only prevent the execution of higher priority tasks as a result of having locked a shared resource.
- The duration of the blocking is bounded by the maximum execution time of an operation on the resource invoked by a lower priority task.

Among the various access protocols that have been proposed for multiprocessor systems, two ex- tensions of SRP have raised a high interest in the context of the Ada programming language: the Multiprocessor Stack Resource Policy (MSRP) [8], and the Multiprocessor resource sharing Protocol (MrsP) [4].

In the rest of the paper the properties of both protocols will be discussed, and their suitability for implementing real-time systems, as well as their relation with the Real-Time Systems Annex [1, D], will be examined. The related subject of protected entry dispatching will also be discussed.

2 Multiprocessor resource sharing protocols

2.1 Multiprocessor Stack Resource Policy

The Multiprocessor Stack Resource Policy (MSRP) [8] extends SRP for multiprocessors. Under this policy, operations on globally shared resources[2] are executed non-preemptively. If an access request by a calling task cannot be immediately granted because the resource is locked by a task running on another processor, the calling task waits on a spin loop until it can gain access to the resource. The spin waiting is also executed non-preemptively. If there are more than one task waiting for the same resource, the requests are served in FIFO order.

The cost of accessing a shared resource is bounded thanks to the FIFO order and the fact that operations on the resource, including spin-waiting if necessary, cannot be preempted. This mechanism ensures that at most one request per processor is issued at a time.

This non-preemptive access policy, however, has the notable drawback of potentially causing arrival blocking to all local higher priority tasks, and not only those with equal or lower priorities than the ceiling priority of the resource.

2.2 Multiprocessor resource sharing Protocol

The Multiprocessor resource sharing Protocol (MrsP) [4] addresses the above described drawback of MSRP by executing the resource operation and the spin-waiting at the ceiling priority of the resource, rather than non-preemptively. This approach still presents the benefits of SRP and, like MSRP, limits the number of concurrent requests from the same processor to one.

However, contrary to MSRP, the access can be interrupted due to a local preemption of the task holding the lock. This interference does not only affect the preempted task, but also remote tasks spin-waiting for the

* This work has been partially funded by the Spanish National R&D&I plan (project M2C2, TIN2014-56158-C4-3-P).

[2] Resources considered as globally shared are those accessed from more than one processor, not necessarily from every processor in the system.

resource, or even future requests, while the lock-holding task remains preempted.

In order to mitigate this effect, MrsP includes an innovative helping mechanism by which a remote task can undertake the access of a locally preempted task. In practice, this is implemented by migrating the preempted task to a remote processor on which a task is spin-waiting to access the same resource. The overhead caused by such migrations can be calculated by analysing the release of local tasks with base priority higher than the resource ceiling priority and the cost of each migration.

An evaluation of this approach shows that in practice MrsP can outperform MSRP, including the migration overhead [16].

2.3 Discussion

MSRP can be implemented, at least partially, using standard Ada and the Real-Time Systems Annex, including the Ravenscar profile restrictions [1, D.13]. Non-preemptive access can be en- forced using Locking_Policy (Ceiling_Locking), by assigning an effectively non-preemptive ceiling priority to every globally shared resource. The protocol does not require migrations, and is thus compatible with the profile and the current Ada 2020 draft [2], which clearly forbids migrations. However, as identified in a previous position paper [5], FIFO queuing and spin waiting are not covered by the current standard. The revised Annex should take care of these limitations, e.g. by extending the queueing policies in D.4 to protected procedures and functions, as these operations can also incur indirect blocking with MSRP. Good practices on active waiting schemes could also be addressed as implementation advices. The Ravenscar profile definition should be updated to clearly define the use of these mechanisms.

The implementation of MrsP poses bigger challenges. While a clean pseudo-implementation is provided in [5], in practice some issues are found, such as the support required for the helping mechanism [6, 15], or support for nested resource policies and transitive helping [6, 11]. It has, however, been shown that this protocol can be used to schedule complex task sets that could not be scheduled under MSRP [10, 16]. Furthermore, while the implementation of MrsP presents a higher complexity, it has been successfully achieved for different RTOS. The research conducted since MrsP was proposed in [4, 5] has shown the feasibility and interest of incorporating MRSP_Locking as new locking policy for general Ada, while a Non_Preemptive_Locking could be the choice for the Ravenscar profile [10]. It could also be used in the extended profile that has been the object of several recent proposals [12, 9, 13].

3 Ada entries

The protocols discussed in the previous section can be used to properly access critical sections, i.e. protected actions in Ada terms, so that mutual exclusion is guaranteed and priority inversion is minimized. Ada protected objects can also have entries, with barriers that only allow the execution of the corresponding protected action when open. Tasks invoking a protected entry with a closed barrier are suspended on a queue associated to the entry. Note that tasks requesting exclusive access to the object are not suspended, and can even wait actively as described above for MSRP and MrsP. The Ada standard also specifies that while there are any pending requests waiting on closed barriers in a protected object, no new access requests to the same object can be granted. This is known as the eggshell model. While the effect of these particularities on the scheduling analysis is out of the scope of this paper, their effects on system predictability are addressed below.

As previously recalled, for uniprocessor systems, the Priority Ceiling Protocol (PCP) and the Stack Resource Policy (SRP) were widely adopted due to their predictability. Systems built using PCP/SRP policies are analysable from the timing behaviour perspective, including protected entries: tasks re-dispatched after being blocked on an entry can be modelled as different jobs of the same task and, since entries are the only blocking actions under PCP/SRP, their priority over other accesses does not affect the proper system behaviour under such policies (there cannot be any subprogram trying to access a resource at the time that an entry is served).

In monoprocessors the so-called proxy model approach has been widely accepted as the most convenient from the runtime implementation and system efficiency perspectives. Furthermore, from the real-time systems perspective, it ensures that the benefits of the Ceiling_Locking policy are preserved. In particular, only one task can cause blocking on a higher priority task, at most during the execution of one protected action. The time needed for barriers evaluation be added to the execution time of each protected action. If an action causes one or more barriers to open, the cost of servicing the queued entries must also be taken into account for the access time and thus on the arrival blocking. It does not, however, cause the execution of any other task in contrast to a self-service approach, where tasks on opened entries are to be dispatched to perform their accesses.

This effect is increased in multiprocessor systems. In these systems, as stated above, not only entry calls but any call can be blocked accessing a PO. This includes the case in which a subprogram call can be blocked due to an entry with an open barrier that is unable to perform its access due to its host processor scheduling. This kind of situations are already addressed by the scheduling protocols definition (avoided in MSRP thanks to the non-preemptive access and alleviated in MrsP thanks to the helping mechanism). From our point of view, the proxy model can better implement the Ceiling_Locking policy and preserve its properties, including the expected behaviour of MSRP and MrsP and thus should be recommended as an implementation advice in the Real-Time Systems Annex.

4 Conclusions

Recent and ongoing research results on Ada use on multiprocessor platforms have been reviewed regarding task synchronization using shared resources. In particular, different aspects of Ada protected objects have been discussed related to the analysis and implementation of systems in Ada under two multiprocessor protocols. Given their relevance, research maturity and interest among the real-time systems and Ada community we argue on the inclusion of the following items in the next language revision:

- Locking_Profile(Non_Preemptive_Locking).
- Locking_Profile(MRSP_Locking).
- Protected object queueing policies.
- Implementation advice on good practices for the active waiting on multiprocessors shared resources.
- Implementation advice on the use of the proxy model for barrier evaluation and entry servicing.

References

[1] ISO/IEC (2012), *ISO/IEC 8652:2012(E): Information Technology — Programming Languages — Ada*.

[2] ISO/IEC (2017), *ARM2x. Ada Reference Manual ISO/IEC 8652:202x(E), draft 12*. Available at http://www.ada-auth.org/standards/ada2x.html.

[3] T. P. Baker (1991), *Stack-based scheduling for realtime processes*, Real-Time Systems, 3(1):67–99.

[4] A. Burns and A. J. Wellings (2013), *A schedulability compatible multiprocessor resource sharing protocol–MrsP*, In Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on, pages 282–291. IEEE.

[5] A. Burns and A. J. Wellings (2013), *Locking policies for multiprocessor Ada*, Ada Letters, 33(2): 59–65.

[6] S. Catellani, L. Bonato, S. Huber, and E. Mezzetti (2015), *Challenges in the Implementation of MrsP*, pages 179–195. Springer International Publishing, Cham.

[7] R. I. Davis and A. Burns (2011), *A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems*, ACM Computing Surveys, 43(4).

[8] P. Gai, G. Lipari, and M. D. Natale (2001), *Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip*, In Proceedings of the 22nd IEEE Real-Time Systems Symposium. IEEE Computer Society.

[9] J. Garrido, B. Lacruz, J. Zamorano, and J. A. de la Puente (2016), *In support of extending the ravenscar profile*, Ada Lett., 36(1):63–67.

[10] [J. Garrido, J. Zamorano, A. Alonso, and J. A. de la Puente (2017), *Evaluating MSRP and MrsP with the multiprocessor Ravenscar profile*, In J. Blieberger and M. Bader, editors, Reliable Software Technologies — Ada-Europe 2017, pages 3–17. Springer.

[11] J. Garrido, S. Zhao, A. Burns, and A. Wellings (2017), *Supporting nested resources in MrsP*, In J. Blieberger and M. Bader, editors, Ada-Europe International Conference on Reliable Software Technologies, pages 73–86. Springer.

[12] P. Rogers, J. Ruiz, and T. Gingold (2015), *Toward extensions to the ravenscar profile*, Ada Lett. 35(1):32–37.

[13] P. Rogers, J. Ruiz, T. Gingold, and P. Bernardi (2017), *A new profile based on Ravenscar*, Ada Europe 2017 Panel on The Future of Safety-Minded Languages.

[14] L. Sha, R. Rajkumar, and J. P. Lehoczky (1990), *Priority inheritance protocols: An approach to real-time synchronization*, IEEE Tr. on Computers, 39(9).

[15] J. Shi, K.-H. Chen, S. Zhao, W.-H. Huang, J.-J. Chen, and A. Wellings (2017), *Implementation and evaluation of multiprocessor resource synchronization protocol (MrsP) on LITMUSRT*, 13th Workshop on Operating Systems Platforms for Embedded Real-Time Applications.

[16] S. Zhao, J. Garrido, A. Burns, and A. Wellings (2017), *New schedulability analysis for MrsP*, In Embedded and Real-Time Computing Systems and Applications (RTCSA), 2017 IEEE 23rd International Conference on, pages 1–10. IEEE.

Proposal for a New Ada Profile for Small Microcontrollers*

Mario Aldea Rivas
Universidad de Cantabria, Spain; email: aldeam@unican.es
Héctor Pérez-Tijero
Universidad de Cantabria, Spain; email: perezh@unican.es

Abstract

This paper presents a proposal for a new Ada profile targeted to microcontrollers with tight memory constraints. The profile has the same restrictions that the Ravenscar profile but includes a new scheduling policy based on the "one-shot task" model that allows stack sharing techniques to be applied to Ada tasks. A preliminary implementation based on a small kernel, a modified run-time system and an automatic code generation tool has been developed. The initial tests bring promising results, showing the profile is functionally correct and has a small memory footprint.

1 Introduction

An important portion of the embedded system market is occupied by small microcontroller units (MCU) with tight memory constraints. According to [1], 8-bit and 16-bit MCUs are the main processor in 21% of the embedded projects, and they are also widely used as secondary processors in many other embedded projects. In 2015, for instance, there were sold three 8-bit microcontrollers for each 32-bit microcontroller [2].

Microcontrollers with tight memory constraints are very popular in industry and among hobbyists. Examples of those popular devices are the Microchip Technology's ATmega328P (the core of the Arduino Uno), which is an 8-bit MCU that features 32KB of Flash memory and 2KB of SRAM, and the SAM series (also produced by Microchip) with an ARM Cortex-M core and only 8KB of Flash memory and 4KB of SRAM in the smallest models.

MCUs are mostly programmed in C, while the use of Ada is marginal in this sector (under 1%) [1]. Despite the limited resources of the small MCUs, there are quite a few Real-time Operating Systems (RTOS) ported to these devices, such as FreeRTOS[1], ERIKA[2] or SMX[3]. All these RTOSs usually provide simple tasking primitives and scheduling policies fitted to these memory-constrained devices.

*This work was partially supported by the Spanish Government under grant number TIN2014-56158-C4-2-P (M2C2).

[1] https://www.freertos.org/

[2] http://erika.tuxfamily.org/

[3] http://www.smxrtos.com/

We consider that the development of software for small MCUs can benefit from the features of the Ada programming language, like programming by contract, strong typing, representation clauses, static compiler checks and advanced tasking primitives among others. At the same time, the availability of Ada implementations for popular devices used by hobbyists all around the world (e.g., Arduino Uno) can contribute to the promotion of the Ada language.

In this paper we explore the possibility of defining a new Ada profile that includes a simple form of tasking implementable in microcontrollers with tight memory constraints.

2 Precedents

There are some Ada implementations for small microcontrollers (specially for MCUs of the AVR family) like the AVR-Ada Project[4] and the AVR GNAT GPL 2012 cross compiler (hosted on Windows). These implementations only provide the "Zero Footprint" run-time, a limited run-time system without tasking support.

The Ada language offers to programmers a rich and powerful set of tasking constructs which require a large run-time system to support them. For example the footprint of an Ada tasking application for MaRTE OS [3] is at least 200KB (this size includes the GNAT run-time and the MaRTE kernel, but not the tasks' stack).

The tasking subset defined in the Ravenscar profile [4].D.13 requires a much smaller run-time system. According to [5] the memory footprint of an Ada application using this profile is around 10KB (excluding stacks). Its size makes Ravenscar suitable for medium size MCUs, but not for the smallest ones mainly due to the difficulty to apply "stack sharing" techniques as we will see later on.

Tasks stacks are large memory consuming resources. For example, the default stack size for the GNAT Ravenscar runtime is 4KB, the minimum stack size in a relatively simple kernel as FreeRTOS is between 200B and 500B depending on the MCU architecture, and similar stack sizes can be seen in other simple RTOS. In consequence, a relative low number of tasks would rapidly exhaust the RAM available in a small MCU.

In order to limit the amount of memory allocated to tasks stacks, RTOSs for memory constraint systems implement

[4] https://sourceforge.net/projects/avr-ada/

mechanisms that allow some kind of "stack sharing" among tasks. Most common stack sharing techniques require that tasks behave as "one-shot tasks", that is, tasks that do not keep any local state in the stack between activations.

Each "one-shot task" is made of initialization instructions, that are executed after the creation of the task, and task job instructions. Task job instructions always should end with a potentially blocking operation to wait for the next activation event. Apart from this, no other blocking operations are allowed in the task job instructions. After each arrival of the activation event, the task job instructions are executed and, according to this model, no local state is stored between task job executions.

Several "one-shot tasks" can share the same stack if they execute in a nested fashion, i.e. a high priority task always ends its job before all the tasks it has preempted. To verify this behavior, tasks can not block during the job (i.e. only blocking at the end of the job is allowed) and an appropriate resource protocol must be used. Examples of such resource protocols are the Immediate Priority Ceiling (when task are scheduled under a fixed-priority policy) and the Stack Resource Protocol [6] (when the scheduling policy is EDF). Even using this approach the size of the stack can be large in the worst case. For example, the maximum stack size for fixed-priority scheduling will be equal to the sum of the maximum stack sizes of all priority levels.

The maximum memory saving can be achieved when "one-shot tasks" are used with a non-preemptive scheduling policy. In such case, the same stack area can be used by all the tasks.

One of the most popular operating system standards for automotive embedded systems, the OSEK/VDX[5] (also included in the AUTOSAR standard), allows stack sharing with its "basic tasks" (which are an example of "one-shot tasks") and the Immediate Priority Ceiling. This standard also allows tasks to be declared as "non-preemptive" to achieve the maximum stack saving when desired.

In the case of the Ada language, it is important to notice that the `Non_Preemptive_FIFO_Within_Priorities` dispatching policy [4].D.2.4 does not allow direct stack sharing among tasks. The reason is that this policy is not enough to implement the "one-shot tasks" model because: (a) tasks status is kept in the stack between activations and (b) tasks can block during the job.

3 Profile requirements

As stated in Section 2, the complexity of the full Ada tasking is excessive to fit in a small MCU, and even the reduced tasking subset defined in the Ravenscar profile is too large for the smallest microcontrollers. For that reason, in this paper we propose the definition of a new Ada profile that can be used in MCUs with tight memory constraints. The requirements for the proposed profile are:

[5] https://www.autosar.org/

Listing 1: One-shot task structure

```
task body One_Shot_Task is
   One_Shot_Task_Local_Data;
begin
    One_Shot_Task_Initialization ;
    Wait_For_First_Activation_Event ; ( optional )
   loop
      -- One shot task job
      One_Shot_Task_Job_Body;
      Wait_For_Next_Activation_Event;
   end loop;
end One_Shot_Task;
```

- Allow the implementation of a simple run-time system with a very small footprint (a size between 5KB and 10KB would be acceptable).
- Define a scheduling algorithm that allows using "stack sharing" techniques by forcing Ada tasks to behave as "one-shot tasks".
- The language syntax should be maintained as close as possible to original language syntax (ideally no syntax changes should be required to support the profile).
- Introduce tasking restrictions identical (or very close) to those imposed by the Ravescar profile.
- A program for this new profile can be compiled using the Ravenscar profile or the full Ada tasking without modifications.

4 Profile definition

The main objective of our proposal is to define a set of conditions which allow Ada tasks to behave as "one-shot task". To achieve so, tasks' body must have a structure similar to the one shown in Listing 1.

Each "one-shot task" is made of a initialization part (`One_Shot_Task_Initialization`) that is executed only once and an endless loop that wraps the code executed in each task job. In turn, the task job is made of the instructions that form the task job body (`One_Shot_Task_Job_Body`) and, as the last instruction, a potentially blocking operation to wait for the next activation event (`Wait_For_Next_Activation_Event`).

Potentially blocking operations are only allowed as final instruction of the task job (`Wait_For_Next_Activation_Event`) and, optionally, as the instruction just before the endless loop (`Wait_For_First_Activation_Event`).

It is worth noting that the task structure shown in Listing 1 is, in fact, a typical structure for an Ada task. However, other task structures allowed in full Ada or in Ravenscar are not allowed in our proposal: no instructions are allowed between `Wait_For_First_Activation_Event` and the beginning of the endless loop nor between `Wait_For_Next_Activation_Event` and the end of the loop.

One task can wait for its next activation on any potentially blocking operation defined in the Ravenscar profile. In Ravenscar, such operations are limited to: a delay until, an entry of a protected object or a language-defined subprogram that is potentially blocking (for example blocking on a suspension object by calling to `Ada.Synchronous_Task_Control.Suspend_Until_True`). Since potentially blocking operations are clearly defined in Ravenscar, the compiler can detect if a task does not follow the expected structure.

As it can be deduced from Listing 1, the task executes its job body after the arrival of any activation event. In order to verify the "one-shot task" model, no local state is present when the task job starts its execution. Local variables can be created in block statements inside the task's job body but these block statements cannot wrap the potentially blocking operation at the end of the loop. In order to keep the Ada semantics, task's local variables (labeled as `One_Shot_Task_Local_Data` in Listing 1) must be created in global memory since they status must be preserved between activations.

Similarly to regular tasks, the main task should have the same structure that the "one-shot task" in Listing 1. Main task's local variables can be created in the stack or as global variables. In case they are created in the stack, at the beginning of the main task body the position of the top of the stack is stored. Stack is restored to that position at the beginning of the execution of the body of any task.

We propose the definition of a new profile (`Stack_Sharing_Ravenscar`) with all the restrictions of the Ravenscar profile plus a new one to force the tasks to obey the "one-shot task" structure discussed earlier in this chapter. The second difference between the proposed profile and Ravenscar is the introduction of a new dispatching policy (`Non_Preemptive_Stack_Sharing`) which behaves equal to `Non_Preemptive_FIFO_Within_Priorities` but defining all potentially blocking operations as a dispatching points regardless if the task actually blocks in the operation or not. This allows to use any potentially blocking operation as the activation mechanism of our "one-shot tasks": once the task ends its job a dispatching operation is performed and the higher priority ready task is chosen to execute.

When the scheduling policy is non-preemptive, there is not need for using any primitive to ensure exclusive access to resources. However, in the proposed profile we include the use of protected objects (PO) for three main reasons: (a) POs use improves code compatibility with full Ada and Ravenscar; (b) POs clarify the code structure since they enclose a set of related data that is intended to be shared among tasks; and (c) PO entries are smart constructs intended to be used as activation points of event-driven tasks.

The GNAT compiler implements POs using the "proxy model" which fits perfectly our "one-shot task" model. In the "proxy model" when, as a consequence of a protected operation, an entry with a queued task is opened, the body of the entry is executed by the task that is executing in the PO. It is only at the end of the entry when the waiting task is activated. Consequently, the activated task can start is new job executing directly its job's body as dictated by our "one-shot task" model.

5 Preliminary implementation

We have developed a preliminary implementation of the Ada profile presented in this paper. It mainly consists on a small kernel, a modified run-time system and an automatic code generation tool:

- The kernel is the piece of code that provides support for the "one-shot task" model and interacts with the hardware. The current version supports three different boards: Arduino Uno, Raspberry Pi 1 and STMicroelectronics STM32F.
- The run-time system is based on the "Small Footprint" Ravenscar run-time provided by the GNAT GPL 2017 cross-compiler for ARM bare boards[7]. In our modified version of the run-time, the original bare board support has been removed and replaced by invocations to the kernel layer. Additionally, a small number of packages have been modified, most of them with minor changes. Mayor modifications have been required in the packages involved in PO implementation and task creation.
- The automatic code generation tool is responsible for adapting the application code to be run on top of the kernel layer, as the one-shot task model imposes some restrictions on the implementation and the structure of Ada tasks. This process of adapting the source code is transparent to the end-user, who can then focus on writing Ravenscar-compliant source code as in other platforms. To analyze and adapt the application code, the tool relies on the Ada Semantic Interface Specification for GNAT (ASIS)[8].

In our initial tests, we have found promising results in relation to the memory footprint. For the Arduino Uno board, a simple application with two tasks and one protected object (one of the tasks is periodic and the other is activated by a PO entry) has a memory footprint of around 6KB, and the maximum stack usage remains under 100B.

6 Conclusions and future work

We have presented a proposal of a new Ada profile that includes a scheduling policy that enables the use of stack sharing techniques. The key idea is to define the structure expected for a task to behave as a "one-shot task". The proposed structure is versatile and similar to the usual patterns for Ada tasks.

The initial tests with a preliminary implementation of the profile indicate that it is functionally correct and the memory footprint of the applications is small enough for the kind of systems it is targeted to.

[7] https://www.adacore.com/community

[8] https://www.adacore.com/documentation/asis-for-gnat-users-guide

Both, implementation and profile definition, require further work. On the implementation side, the run-time system adaptation must be completed. The main aspect pending to be implemented is the support for interrupt handlers as protected procedures.

The profile definition could be extended in order to reduce the detrimental effect of non-preemption on the schedulability of the system. In this line, two alternatives are under consideration, both implying a compromise between stack sharing and responsiveness:

- Cooperative scheduling: allow tasks to be preempted when they call a yield or a relative delay statement.
- Preemption thresholds [7]: this strategy allows to limit the tasks able to preempt other task to those with priorities above a given threshold.

References

[1] EETimes and Embedded.com (2017), *Embedded Markets Study Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments*, Technical report, 2017.

[2] Wayne Freeman (2016), *11 Myths About 8-Bit Microcontrollers*, Technical report.

[3] M. Aldea Rivas and M. González Harbour(2001), *MaRTE OS: An ada kernel for real-time embedded applications*, Craeynest D., Strohmeier A. (eds) Reliable SoftwareTechnologies — Ada-Europe 2001. Ada-Europe 2001. Lecture Notes in Computer Science, 2043:305–316.

[4] ISO/IEC (2016), *Ada Reference Manual, ISO/IEC 8652:2012(E) with COR.1:2016*.

[5] José F Ruiz (2005), *GNAT pro for on-board misson-critical space applications*, Vardanega T., Wellings A. (eds) Reliable Software Technology – Ada-Europe 2005. Ada-Europe 2005. Lecture Notes in Computer Science, 3555(17360):248–259.

[6] T.P. Baker (1990), *A stack-based resource allocation policy for realtime processes*, In Proceedings 11th Real-Time Systems Symposium, pages 191–200. IEEE.

[7] Yun Wang and M. Saksena (1999), ‚*Scheduling fixed-priority tasks with preemption threshold*, In Proceedings Sixth International Conference on Real-Time Computing Systems and Applications, pages 328–335. IEEE Comput. Soc.

Ravenscar-EDF: Further Results from Improved Comparative Benchmarking

Paolo Carletto and Tullio Vardanega
Department of Mathematics, University of Padua, IT-35121

Our project was motivated by the sense of incompleteness in the EDF-to-FPS single-processor scheduling comparison reflected in the relevant literature [1], [3]. What we deemed most unsatisfactory was that, while [1] ignored runtime costs altogether – a common trait of theoretical works –, [3] addressed performance considerations making somewhat arbitrary assumptions on the implementation solutions.

To pursue our goal of a more grounded comparison, we created a fork from AdaCore's original implementation of the Ravenscar Profile FPS runtime for the Leon processor (which "industrialized" UPM's ORK [2]). To seek maximum fairness, our fork simply replaced the FPS components of the runtime, including support for the Immediate Priority Ceiling Protocol (IPCP), with the equivalent EDF components, which incorporated support for the Deadline Floor Protocol. Every other part of the runtime was left unchanged.

We created a test suite running on a trial version of the TSIM LEON Simulator by Cobham Gaisler, which yielded a good set of benchmark data that helped us to experimentally confirm some held beliefs in the EDF-to-FPS comparison, and to confute others.

Those experiments however suffered two important limitations, which we set out to remove in the second, unpublished, stage of our project. First, we removed the bounded-duration constraint of the execution runs (attached to the trial version of the simulator) so as to remove any artificial constraints on the length of the hyperperiod of the synthetic task sets included in our test suite. Second, we implemented more sophisticated event capture mechanisms to enable more accurate discrimination between single events and recurring behaviour, especially with regard to preemptions.

The data set we are capturing with our benchmarks should enable us to contribute experimental evidence to answer the question of how does the EDF runtime performance relate to that of FPS, from the concreteness of an implementation perspective.

References

[1] L. Liu and J. W. Layland (1973), *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*, Journal of the ACM, 20 (1): 4661.

[2] J. A. de la Puente, J. Zamorano and J. Lopez (2009), *GNATforLEON 2.1.0 ⁄ ORK+ 2008,* Universidad Politécnica de Madrid (UPM).

[3] G. Buttazzo (2005), *Rate Monotonic vs EDF: Judgment Day*, Real-Time Systems, 29, 5-26.

Ravenscar Support for Time-Triggered Scheduling*

Jorge Real
Instituto de Automática e Informática Industrial, Universitat Politècnica de València, Spain; email: jorge@disca.upv.es
Sergio Sáez
Instituto Tecnológico de Informática, Universitat Politècnica de València, Spain; email: ssaez@disca.upv.es
Alfons Crespo
Instituto de Automática e Informática Industrial, Universitat Politècnica de València, Spain; email: alfons@disca.upv.es

Abstract

This position paper follows from a previous proposal to integrate a time-triggered scheduler in a priority-based, preemptive scheduler such as that supported by Ada's task dispatching policy FIFO_Within_Priorities. *The resulting combined scheduling carries the advantages of both time-triggered and priority-based scheduling, and helps mitigating their drawbacks.*

The paper presents a system model for the time-triggered subsystem that extends the original proposal, and describes a Ravenscar implementation of the scheduler at the run-time system level, in the form of a new package Ada.Dispatching.TTS. *Multiple programming patterns can be implemented on top of this scheduler. With respect to the previously proposed full-Ada implementation, only patterns that implied the use of asynchronous transfer of control have been excluded. On the other hand, the extension of the original model enables new patterns, not supported in our previous proposal, using the new types of continuation and optional slots.*

We hold that bringing the time-triggered paradigm to Ravenscar is both feasible and convenient for the High-Integrity and Embedded application domains.

Keywords: Real-Time Systems. Time-Triggered Scheduling. Priority-Based Scheduling. Ravenscar Profile. High-Integrity Systems. Embedded Systems

1 Introduction

In the real-time systems domain, there are two major approaches to scheduling real-time tasks. One is time-triggered scheduling (TTS) whereby each task is executed during the exact time intervals dictated by a fixed plan designed in advance. The other is priority-based scheduling (PBS), in which the time intervals when a task executes are decided at run time, based on the task's priority. The priority attribute of a task can in turn be static or dynamic, leading to several variations of the idea.

*This work has been partly supported by the Spanish Government's project M2C2 (TIN2014-56158-C4-1-P-AR) and the European Commission's projects ENABLE-S3 and AQUAS (ECSEL-JU, Contracts 692455 and 737475).

There are pros and cons to both approaches. While TTS is superior to PBS in terms of predictability and reduced jitter (i.e., precise and rapid task release), a time-triggered schedule is difficult to design for non-trivial cases – actually, it is an NP hard problem. Another interesting property of TTS systems is that access to shared resources is simpler, provided all tasks execute non-preemptively; whereas PBS preemptive systems require mechanisms to enforce mutual exclusion in the access to shared resources. A fundamental advantage of PBS over TTS is that it keeps a convenient separation of concerns between timing and functional requirements, making PBS systems easier to modify and maintain.

In previous papers we have proposed an architecture supporting the execution of applications that include a mix of TTS and PBS workload [7, 6]. The idea was to divide the application into two subsets of tasks, one scheduled according to a time-triggered plan, and the other one scheduled by a priority-based scheduler. The implementation of this architecture reserves the highest priority of a priority-based scheduler to the TTS subset of tasks, and lets the PBS subset execute during the spare time left by the time-triggered workload, using the rest of priority levels. The time-triggered schedule is driven by an Ada timing event handler (like a timer interrupt handler), which allows the system to react promptly to the arrival of time events, hence keeping a reduced release jitter for TTS tasks.

Combining both scheduling approaches helps mitigating their drawbacks and taking advantage of their benefits. For example, the time-triggered load can be reduced to just the set of more jitter-sensitive tasks (such as control or communication tasks) and the rest of tasks (logging, user interface, optimisation tasks,...) can benefit from a priority-based scheduler. This mitigates the difficulty of designing the TT schedule, since there are less tasks to consider. For the rest of application tasks, scheduled by the PBS, the designer is relieved from the burden of building an offline schedule for them. The price to pay with this combined scheduling approach is that, ultimately, the underlying scheduler is a PBS, which cannot be as efficient as a TTS since it has to execute context switches between tasks.

In summary, combined scheduling (TTS-PBS) gives good results for jitter control, slot-based communication and reuse of pre-designed TT schedules. Our aim in this paper is to adapt the technique proposed and implemented in [7, 6] to Ravenscar, towards facilitating the adoption of this approach in High-Integrity, certifiable, real-time and embedded systems, which is the target niche of the Ravenscar profile.

The rest of this paper is organised as follows. Section 2 describes the system model with regard to the time-triggered subset only, given that our model does not modify the fixed-priority, preemptive

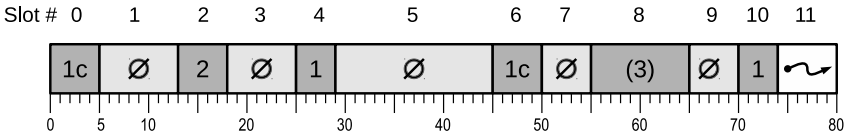


Figure 1: A 12-slot time-triggered plan. Slots marked 2, 4 and 10 are regular slots for works 1 and 2, as indicated; slots 0 and 6 are continuation slots for work 1; slot 8 is an optional slot for work 3; and slot11 is a mode change slot. The rest are empty slots.

scheduling model already supported by the Ravenscar profile. Section 3 proposes a variety of behavioural patterns for time-triggered works that must be executed according to the TT plan, and in Section 4 we describe design and Ravenscar implementation aspects of the TT scheduler. We present some experimental results in Section 5, focusing on jitter measurements obtained from running the TT scheduler on an embedded board using ARM's STM32F407VGT6 micro-controller unit. We finally present our conclusions in Section 6.

2 System Model: the Time-Triggered Plan

Our system model combines two disjoint subsets of tasks: (*i*) a subset scheduled according to an offline, static time-triggered plan; and (*ii*) another subset of tasks scheduled under a priority-based, preemptive scheduler. The two subsets run on a common priority-based, preemptive scheduler, but the time-triggered workload always takes precedence over the priority-based subset. Due to this reason, the set of priority-scheduled tasks does not interfere the execution of the TT plan.

Since we are aiming at implementing our model with the Ravenscar tasking profile restrictions, the priority-based subset can only be scheduled based on a fixed-priority scheme such as Rate Monotonic or Deadline Monotonic [4, 3]; but nothing in our model precludes the use of dynamic priority algorithms such as EDF [4] for this second subset, or even a combination of schedulers using different priority bands. For this reason, we will limit ourselves to describe the system model for the time-triggered plan.

A time-triggered plan is a cyclic sequence of actions to be executed at particular points in time. The plan is described by means of an ordered list of *time slots*, each of its own *slot duration*. If a slot starts at time t, its lifetime goes from t to $t+$*Slot Duration*. There are no gaps between slots: each slot starts just at the end of the previous slot in the plan. In other words, the duration of the plan is the sum of slot durations.

Figure 1 shows a 12-slot example plan, with slots numbered from 0 to 11. Apart from its duration, each slot is characterised by the type of actions to take during the slot lifetime. Each slot has a particular mark indicating its type (digits and other symbols whose meaning will be described in a moment). Using the time scale in milliseconds at the bottom of the plan, it can be seen that the plan has a duration of 80 ms, slot 0 has a duration of 5 ms, slot 11 takes 6 ms from time 74 ms to 80 ms, etc.

There are five possible types of slots in a plan:

- A ***regular slot*** defines a time interval reserved for the execution of a time-triggered task (a *work*). It is denoted by a *regular Work_Id*, a strictly positive integer value that identifies the particular work to execute during the slot. In Figure 1, slots 2, 4 and 10 are regular slots corresponding to works 1 or 2 as indicated. The underlying TT scheduler will make the work start to execute as soon as feasible after the start time of the slot.

  The duration of a regular slot must be sufficient, by design, to accommodate the worst-case execution time of the work it serves. If a work overruns its regular slot then the scheduler will resort to raising a Program_Error exception, since an overrun violates the schedulability assumptions of TTS. If, on the contrary, a work completes before the end of the slot duration, then the rest of the slot remains available for PB tasks. A TT task must always be ready to use its allocated regular slots in the plan. Failing this, the scheduler will raise Program_Error as well. The next type of slot is more permissive in this regard.

- An ***optional slot*** is like a regular slot except that it can be omitted. A TT task may decide to use or not to use an assigned slot in the plan. If it does use it (the task is ready to start when the optional slot starts) then it has the same semantics as a regular slot, including overrun control at the end of the slot. But if the task is not ready at the start of the slot, it is not considered an error and the slot duration is made available for PB tasks. In Figure 1, slot 8 is an optional slot for work 3, indicated with parentheses.

  Optional slots are useful for tasks that may or may not require to use their allocated slot, such as a communication task when it has nothing to say; or a sporadic task whose activation event has not occurred.

- A ***continuation slot*** can be regarded as a special kind of regular slot, in the sense that it is associated to a particular Work_Id. In Figure 1, slots 0 and 6 are continuation slots. They are marked with a regular Work_Id plus a letter 'c', indicating continuation.

  What is special about these slots is that the work they host does not need to be completed by the end of the slot; it can be continued in future slots. Failing to finish by the end of a continuation slot is not considered an overrun. Instead, the work is held at the end of the slot and resumed at the start of the next slot marked with its Work_Id. There may be a number of consecutive continuation slots for a given work. Overrun will only be checked when the plan reaches a regular slot for this work. We will refer to the last, non-continuation slot of a series, as a *terminal slot*. In Figure 1, slots 4 and 10 are terminal slots for work 1, given that they are preceded by continuation slots 0 and 6 for work 1, respectively.

  This type of slots are useful to split a large time-triggered task into smaller pieces in a way that is essentially transparent to the task code. We will visit this pattern in more depth in Section 3. Continuation slots require asynchronously holding and resuming a running task, which in turn requires support from the runtime system. This is the reason why our implementation of the TT scheduler (Section 4) is an extension of the runtime system, in the form of a new package Ada.Dispatching.TTS. The hold/release mechanism is indeed to be taken very carefully, specially with regard to its interaction with protected

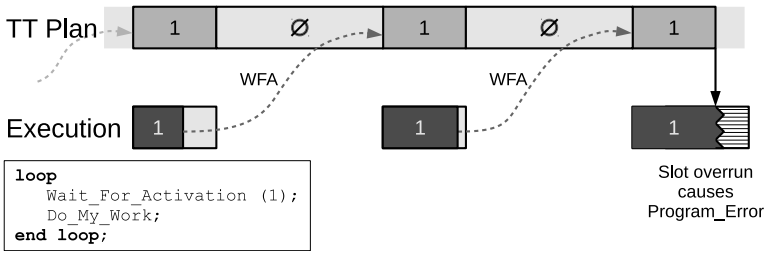


Figure 2: Simple TT Pattern. The TT task uses Work_Id = 1. It just waits for the start of the next slot with that Work_Id and then executes its application code.

actions[1]. But it is doable under certain restrictions as we will show later.

The following two types of slots correspond to scheduler actions exclusively and they have no associated time-triggered task to execute.

- An ***empty slot*** defines a time interval during which no TT activity is planned. This is useful for inserting gaps in the plan where they are needed, making the CPU available to priority scheduled tasks. Note that, even though there is no application-specific activity to execute during an empty slot, there will be scheduler actions executed at the beginning of the slot, as described in subsection 4. In Figure 1, slots 1, 3, 5, 7 and 9 are empty slots.
- A ***mode-change slot*** is similar to an empty slot in the sense that it has no associated work to execute. But additionally, it defines the times in the plan where it is possible to substitute the current plan with a new one. By placing mode change slots in the plan, the designer determines the points where the system can admit a mode change. At the start of a mode-change slot, the TT scheduler will check whether there is a pending mode-change request to process. If there is one, then the new plan will start executing at the end of the mode-change slot. The change will be immediate if the mode-change slot duration is defined to be zero. The ability to change mode (substitute the current plan with a new one at run time) introduces a degree of flexibility that off-line, static schedules do not possess by nature. In Figure 1), slot 11 is a mode change slot, indicated with a curved arrow.

For comparison with the TT plan model we defined in previous papers [7, 6], the model we have just defined introduces the new types of continuation and optional slots. The former are motivated by feedback received from participants at the 18th International Real-Time Ada Workshop suggesting that "*[...] one should be able to divide a long-running time-triggered task into segments that would be executed across several slots [so that] spreading the TT task execution across several slots [would give] chances for other tasks to execute in between these slots.*" [5]. With this type of slots we want to give support to this concept, although it has relevant implications that we will present in Section 3, in the context of patterns using this type of slots.

Finally, note that in this model we are assuming that a plan is executed on a single CPU: there are no overlapping slots. This assumption will help us keep the rest of this paper as simple as possible. However, note also that in a multiprocessor system, the model is applicable provided that it is fully partitioned, i.e., there is only one plan per processor and tasks are statically fixed to CPUs. With careful synchronisation of plans, it is also conceivable to allow data sharing between tasks running in different CPUs. Beyond this restrictive setup, we have also suggested to use controlled forms of migration between plans, so that a task can alternate slots in plans on different processors, to balance the overall TT workload [7]. But we will assume a single processor platform for the rest of this paper, except when explicitly mentioned otherwise.

[1]This difficulty alone can explain the general lack of support for the standard package Ada.Asynchronous_Task_Control

3 Time-Triggered Task Patterns

The model described in the previous section grants time slots for the execution of TT tasks, leaving time gaps to be used at lower priority levels by PB tasks. The ability to use both regular and continuation slots, opens the possibility to define a number of behavioural patterns for the TT tasks using them. This section proposes a set of such patterns. From some of these patterns we will derive requirements for the design and implementation of the TT scheduler that will be presented in Section 4.

3.1 Simple TT Task Pattern

The simplest pattern we can think of is a TT task that accommodates all its execution time within the duration of one slot. The task structure is simply an infinite loop where the task waits for the arrival of its next slot and then executes its sequence of statements, just before suspending itself again until the arrival of its next slot.

Figure 2 represents an example of this pattern, showing the execution of three iterations of a simple TT task. The task uses Work_Id = 1 (for example) in a call to the scheduler to wait for the arrival of the next slot marked with this Work_Id (Wait_For_Activation(1) in the example). Slots assigned to a simple TT task must be all regular slots. At the beginning of each slot, the scheduler releases the work and lets it run at the highest priority among all application tasks. The priority-based subset is therefore disabled to run, since it must use strictly lower priorities than TT tasks. When the task completes within the slot duration (first and second cases of the example in Figure 2), it is suspended by a new call to Wait_For_Activation. The time not used by the TT task becomes available for the lower-priority PB tasks.

If, for whatever reason, the execution time of a simple TT task exceeds the slot duration, this is considered a hard deadline violation and Program_Error is raised. The TT scheduler is thus in charge of making this check at the end of regular slots.

A simple TT task may have its own local state, which is kept across successive releases. It can also share data with other simple TT tasks, because this type of task executes in mutual exclusion with other simple TT tasks (there are no overlapping slots). If the task needs to share data with preemptable PB tasks (or sliced TT tasks, as we'll see later), then it needs to do it via protected objects. In such case, the TT task may experience blocking that must be taken into account when deciding the slot duration.

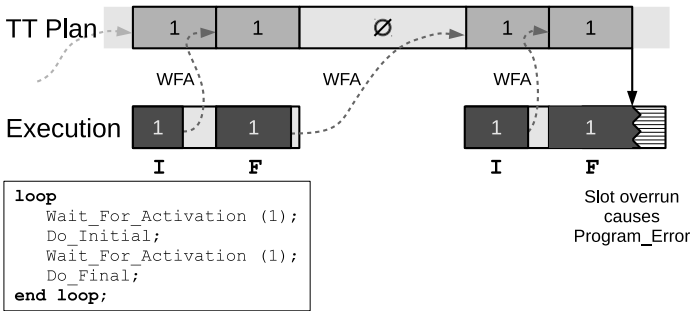


Figure 3: Initial-Final Pattern.

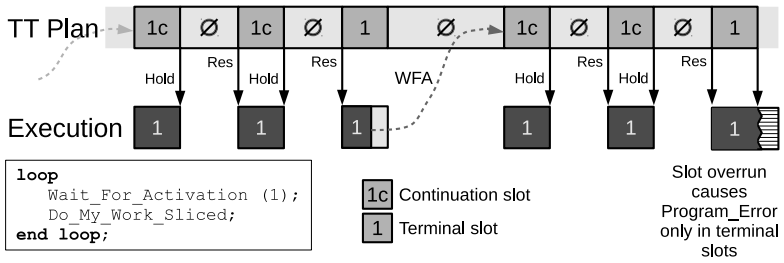


Figure 4: Sliced Task Pattern. Like a simple TT task, it just calls Wait_For_Activation and then performs its work. But execution of the work can be split accross several consecutive continuation slots.

3.2 Initial-Final Pattern (I-F)

The Initial-Final pattern (I-F, for short) is for TT tasks that can be subdivided in two parts, both with strict jitter requirements. This pattern can be easily obtained by sequential composition of two simple TT patterns, as in Figure 3, which shows the execution of two iterations of an I-F task. The loop is split in two parts, first the initial and then the final, and both use the same Work_Id when calling Wait_For_Activation – not necessarily as a restriction, but just to keep the plan more human-readable. Note that the slots for the initial and final parts need not have the same duration. Overrun must be checked for both parts.

Regarding data sharing, the considerations we made about simple TT tasks apply also to the case of an I-F task. Furthermore, communication between the initial and final part is straightforward, since they are both the same task.

3.3 Initial-Mandatory-Final Patterns (I-M-F and I-{M}-F)

This pattern (I-M-F, for short) uses three consecutive regular slots to perform a logically related sequence of TT actions. This scheme is typical in embedded control systems, where the initial part acquires some environment data, the mandatory part does some calculations with the acquired data, and the final part applies the results of the mandatory part to actuators of the controlled system.

The logical structure is defined by three calls to Wait_For_Activation using the same Work_Id, preceding the statements of the initial, mandatory and final parts. The same considerations regarding overrun detection and data sharing we made for simple and I-F tasks, apply to I-M-F tasks: overrun is checked for each and every part of the task.

Actually, this pattern can be generalised to a form I-{M}-F, where there are one or more slots dedicated to execute parts of the mandatory section, each part with overrun control. If we do not want overrun control in all the mandatory slots, then we need to use continuation slots, as the following patterns do.

3.4 Sliced TT Task Pattern

This pattern allows one to break a long running TT task into slices in a way that is transparent to the application code, i.e., it does not require the task to make successive explicit calls to Wait_For_Activation at particular points of its execution. Slicing is dynamic and occurs at run time, rather than statically hardcoded. The TT scheduler will stop and restart the task at points dictated by the plan. A sliced TT task requires the use of one or more continuation slots, ending with a terminal, regular slot.

Figure 4 shows two iterations of execution of a sliced TT task. This task can make use of up to three consecutive slots, which is reflected in the plan as two continuation slots (marked '1c') and one terminal, regular slot (marked simply '1'). The task structure does not differ in structure from the simple TT task described in Section 3.1, but the semantics are totally different due to the use of continuation slots. In other words, one needs to look at the plan to distinguish a simple TT task from a sliced TT task.

In the two iterations shown in Figure 4, the task is held (by the TT scheduler) at the end of exhausted continuation slots, and resumed at the start of its next slot in the plan. Exceeding the lifetime of a continuation slot is not an overrun situation. In the first iteration, the task completes its work within its last slot (a regular slot). In the second, it does not finish by the end of the regular slot, hence the scheduler raises a Program_Error exception.

These two cases are relatively simple to consider, but we need to look into more possible situations. Given the pattern structure, the task will call Wait_For_Activation as soon as it is done with the Do_My_Work_Sliced sequence of statements. This may well happen during a continuation slot, before the terminal slot of the sequence. The task *may* use up to three slots to complete, but it could take less. If that is the case, then the scheduler needs to *ignore* the pending call to Wait_For_Activation until the first slot of the next sliced sequence. An early wait for activation must be therefore be *propagated* until the next continuation slot after the next terminal slot, i.e., the next start of a sliced sequence.

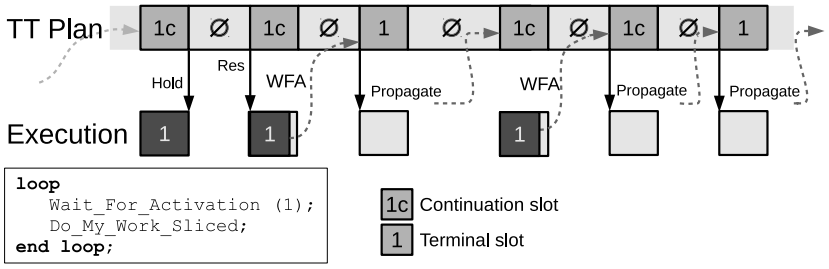


Figure 5: Two iterations of a three-slot sliced task completing before the terminal slot, requiring propagation of early Wait_For_Activation calls.

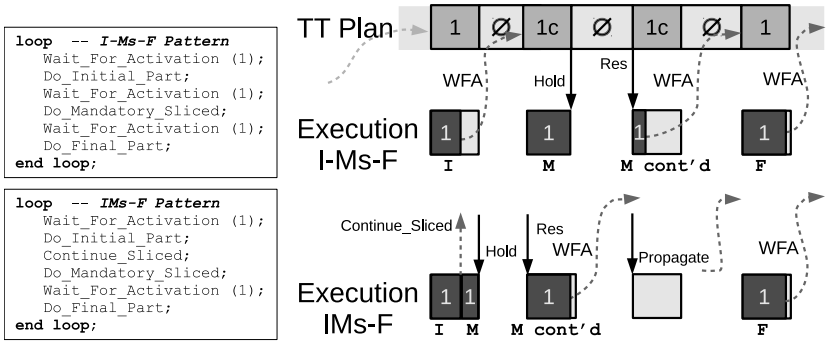


Figure 6: Two variants of tasks with a sliced mandatory part: I-Ms-F and IMs-F.

Figure 5 shows the possible cases of early completion of a three-slot sliced task. In the first case, the task completes in the second slot of a three-slot sequence. When the terminal slot of this sequence arrives, the scheduler has to avoid waking up the task at the start of the slot and checking for overrun at the end. The effect of Wait_For_Activation must be postponed and the task must remain blocked waiting for the start of a new sequence of slots. This is marked as "Propagate" in Figure 5. In the second case, the task completes even earlier, during the first slot of the sequence. The effects of Wait_For_Activation must be propagated to the next two slots. A new sequence of the sliced TT task starts after the next regular (terminal) slot.

As indicated previously, the need to hold and resume a running task has implications that must be taken into account. The problem is specially relevant if the sliced task shares data with other TT tasks or PB tasks. Since the task can be held asynchronously, this data sharing can only be protected. But holding the task while it is running a protected action is not acceptable, and letting it finish a long protected action could enlarge the release jitter of the next slot. To mitigate the effects of these two issues, at a cost, there are two design aspects to consider:

1. A sliced task can only share data with other TT or PB tasks by means of a protected object. To avoid holding the task while it is running a protected action, the ceiling priority of such protected object must be set at a level that effectively disables interrupts. This is the only way to avoid the execution of the (interrupt-driven) TT scheduler while a sliced task is executing a protected action.
2. As a consequence of the previous point, protected actions involving a sliced TT task must be as short as possible. Typically, they should only involve word-sized data exchanges and perhaps a simple condition evaluation. As few cycles as possible, because we are meanwhile blocking interrupts. If the protected action cannot be so short, then there are still alternatives. One is to design the plan so that all continuation slots are followed by empty slots of sufficient duration to absorb the potential blocking time of the runtime system. If this is not possible, because the data exchange required is large, then it is still possible to make use of multiple buffering techniques in order to reduce the need for mutual exclusion to just the time to swap a pointer.

A final consideration with continuation slots and their use by sliced TT tasks has to do with mode changes: they are not allowed in the middle of sliced sequences, because track of an ongoing sliced sequence may be lost when switching from one plan to another. This is easy to avoid by design, because the plan has to determine the exact points when mode changes may occur, by explicitly including mode change slots.

3.5 Initial-Mandatory_Sliced-Final Patterns (I-Ms-F and IMs-F)

The I-Ms-F pattern is a variant of the I-M-F pattern (as described in Section 3.3), where the mandatory part is sliced (as in Section 3.4). The IMs-F pattern (note the missing dash between the 'I' and 'M' parts) is a slight modification of the former that allows the mandatory part to start executing immediately after the initial part, without waiting for the next slot in the plan. Both patterns have the same representation in the plan, taking one regular slot for the initial part (so that it is subject to overrun control), then several continuation slots for the mandatory part, and one regular slot for the final part.

Note that the IMs-F pattern requires specific support from the TT scheduler. Since IMs-F allows the mandatory sliced part to start as soon as the initial part is done, during the first regular slot, we are effectively transforming the semantics of this particular regular slot into that of a continuation slot. The scheduler must therefore be informed of the termination of the initial part so that, if the initial part is not done by the end of the slot, then there is an overrun; but if it has completed, then the slicing regime has started and the

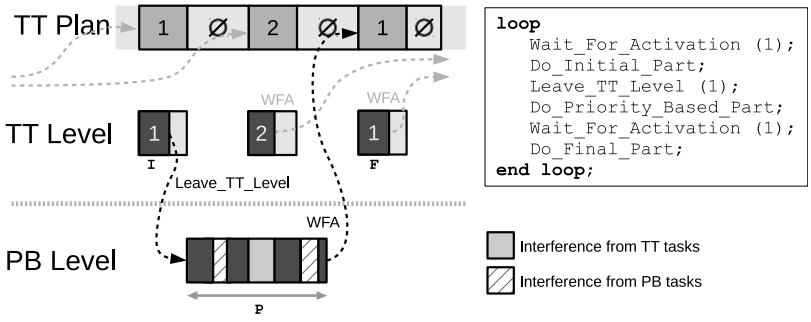


Figure 7: Example pattern of a task with a non-TT part: I-P-F (Initial-Priority_Based-Final)

hold/resume mechanism has to apply to the already started sliced mandatory part.

Figure 6 shows these two patterns for a sliced mandatory part of two slots. The top line represents the TT plan, common to both patterns. The structures of the patterns are shown to the left. The middle row represents a normal execution of an I-Ms-F task. After completing the initial part before the end of the first slot, the task waits for the next activation, hence delaying the start of the mandatory part to the next slot. Since the first slot is regular, the initial part runs under overrun control. The sliced mandatory part takes the next continuation slot and a part of the second continuation slot and then waits for the arrival of the terminal slot, which it uses to execute the final part. In the IMs-F pattern in Figure 6, use of this scheduler service is represented by the call to Continue_Sliced. If the scheduler has not received this call by the end of the first slot, then the initial part has overrun; otherwise, the initial part was completed during the first slot and the running task is held/resumed as an sliced subsequence of this pattern. The final part of the pattern requires a previous call to Wait_For_Activation, as already described for other patterns with a final part.

3.6 TT Patterns with Non-TT Parts

More than just one particular pattern, although we will be using one as an example, this section introduces a scheduler mechanism that makes it possible for a TT task to include parts that are executed in the PB level, in competition with the priority-scheduled tasks subset. The mechanism (Leave_TT_Level) allows a TT task to abandon the TT level and continue execution under the PBS regime. This is useful to execute parts that are not subject to strict jitter requirements and that may be difficult to integrate in the TT plan.

As an example, consider a control task with jitter-sensitive initial and final parts. These parts are used for reading the plant state and for sending commands to actuators, respectively. After reading sensors, the initial part rapidly calculates a first approximation to the control output, to be later applied during the final part. Until that time arrives, an intermediate part tries to improve this calculation by means of an optimisation algorithm that may take disparate execution times, depending on changing environment conditions (e.g., the number of objects detected by a radar). If this middle part had to be included in the TT plan, then the plan would have to provide sufficient slots for the worst-case execution of the optimisation algorithm. But if we could execute this optimisation part as any other PB task, with a selected priority below the TT level, then it would not require slots in the plan, hence keeping it simpler. At the end of the middle part, the task would go back to the TT level to execute the final part with minimal jitter and using the best output possible in the available time.

Figure 7 shows the execution of such *Initial-Priority_Based-Final* pattern, or I-P-F. The pattern requires just two regular slots in the plan for the initial and final parts. In the figure, there is a regular slot for another, unrelated work (Work_Id = 2) in between these two slots of the I-P-F task, which uses Work_Id= 1. The initial part executes during the first slot and, when completed, issues a call Leave_TT_Level to inform the scheduler that the task continues with the execution of the non-TT part at a priority in the PB region. From that moment on, the PB part continues in competition with higher-priority PB tasks and other TT tasks, such as that with Work_Id = 2. The PB part eventually completes with a call to Wait_For_Activation, which makes the task return to the TT level and wait for a slot to execute the final part.

The implementation of the Leave_TT_Level mechanism requires changing the priority of the TT task at runtime, which, at first sight, appears to be in contradiction with the Ravenscar model of fixed priorities. However, a Ravenscar runtime has to actually support a limited form of dynamic priorities, because it is needed to implement the Ceiling_Locking policy for protected objects. Handling PB parts as required by Leave_TT_Level can be supported as well in Ravenscar. Without going into the implementation details that we will visit in Section 4, the problem of scheduling a task with TT and PB parts can be seen as if the task had a base priority in the PB level, at which it runs its PB phase, and an active priority at the TT level when it runs in a TT slot. The mechanism does not need to change the priority of a task other than the running task, as for Ceiling_Locking case. And the changes between base and active priorities occur only as a result of statements executed by the same task that is affected by the priority changes, as is the case for protected actions..

The Leave_TT_Level mechanism can also be used to compose other interesting patterns. For example, a periodic PB task with one TT phase, to be executed during a regular slot in the plan. This slot could be used to synchronise the task with the arrival of slots in the plan, for mutually exclusive communication or data exchange with other tasks, or for accessing a shared resource in general, such as in a slot-based communication protocol.

The P-[F] pattern described now, is another pattern using non-TT parts and that also serves to illustrate the use of optional slots. This pattern models a periodic, PB task, that may or may not use a TT regular slot, for example to synchronise or timely communicate with other TT or PB tasks. At the TT level, the task requires just a sequence of optional slots conveniently spread across the plan. At the PB level, the task executes as any other priority scheduled task.

Figure 8 shows three full iterations of a P-[F] task with a periodic PB part. In the first iteration, the task completes the priority-based part and ends up calling Wait_For_Activation, because the boolean Needed happens to be True. As a result, the final part is executed at

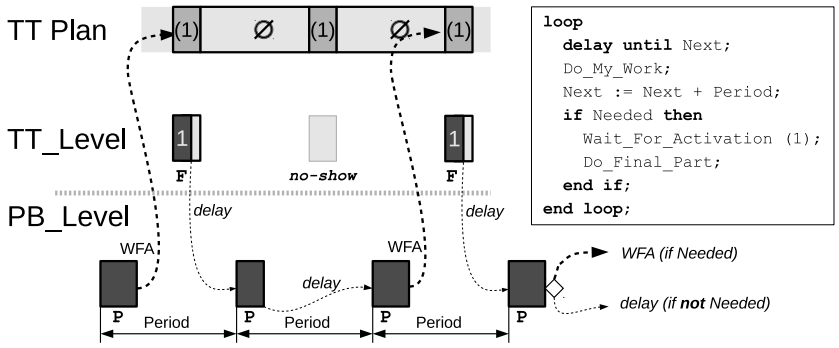


Figure 8: The P-[F] pattern (Priority_Based-Optional_Final) combines the use of non-TT parts and optional slots.

the TT level at the start of the next (optional) slot for this work. In the second iteration, Needed is False and hence the task skips the call to Wait_For_Activation and re-enters the loop to execute the delay sentence instead. Consequently, the task skips its next slot in the plan. If the slot was regular, then this no-show situation would end up in Program_Error. But because the slot is optional, the scheduler knows that this absence of a task waiting for a just started slot is intended and taken care of by the application. If required by the application, synchronisation between time bases of the PB and TT parts can be provided by a similar pattern that reads the clock during the TT part to obtain a new value for Next.

3.7 Patterns: Looking back

As we mentioned in the introduction, our aim with this work was to revisit our previously proposed model and implementation of combined TT-PB scheduling [7, 6]. The goal was to make our implementation Ravenscar-compatible, and hence making it susceptible for consideration in the High-Integrity domain. Given that Ravenscar is a restrictive subset of the tasking model of Ada, one could think that, *a priory*, something will need to be sacrificed. We have found that this almost not true.

Looking back at the patterns we proposed in [7, 6], the only mechanism we have not included is the self cancellation mechanism, which requires the use of Ada's asynchronous transfer of control, a feature that is (wisely) absent in Ravenscar, because it is not an adequate feature in a High-Integrity context, due to the indeterminism it introduces. So this is a sacrifice that stems logically from our new context assumptions.

But it is to be noticed that, despite the Ravenscar restrictions, we have been able to replicate all the functionalities provided by the full-Ada scheduler —the cancellation mechanism mentioned above was implemented by the TT tasks, not the scheduler—. In addition, we have extended the model to include continuation and optional slots, which suggest a number of new patterns.

4 Design and Implementation Details

The time-triggered plan dictates the actions to be taken by the TT scheduler at the start of each slot. The data structure that represents the plan is an array of slot descriptors, each with one field to indicate the slot duration and another three fields to characterise the slot as follows:

- Work_Id - This field contains either a positive value identifying a TT task, or an indication of *empty slot* or *mode change slot*, two reserved non-positive values.
- Is_Continuation - A boolean that marks the slot as a *continuation slot*.
- Is_Optional - A boolean indicating whether the slot is an *optional slot* or not.

Another important source of information for the scheduler is the status of all TT tasks. The scheduler uses the work status to determine the actions to be taken during a slot switch. The status of a TT work is determined by the following boolean fields of a Work_Control_Block record:

- Has_Completed - Indicates if a single-slot work or an sliced work does not require more time at TT level.
- Is_Waiting - Indicates whether the work task is waiting for a new slot or not. This flag is set to True when the work calls Wait_For_Activation.
- Is_Sliced - When this flag is True, it means that this work is currently running sliced, hence it may need to be held/resumed. This flag is set to False when the work is at the start of a terminal slot.

In our implementation, the TT scheduler is the handler of a timing event that is set to trigger at the start of each slot in the plan. Hence it executes at the highest interrupt priority (ARM D.15 12/2 [2]). Based on the slot and work descriptors, the scheduler decides the actions to take during a slot switch. Table 1 describes some of these actions, limited to the case of regular slot processing, for short.

The top part of Table 1 lists two cases of actions to take at the end of a slot. These are *Hold task*, to hold the running TT task when it exhausts a regular slot and must continue sliced in future slots; and *raise Program_Error* when overrun is detected, i.e., the task has not completed and it is not running sliced. To support Hold, our implementation of Ada.Dispatching.TTS uses runtime operations to suspend and extract from the ready queue the low-level thread behind the TT task. This is why a sliced part can only use protected objects with priority ceiling at the highest interrupt priority, as mentioned in Section 3.4.

The bottom part of Table 1 lists scheduler actions related to the immediately starting slot. The actions that are common to most cases in this table (denoted *CA*) are to mark the work as sliced when it enters a continuation slot, and to set the timing event handler to the end of this starting slot. Transferring the Is_Continuation property of the slot to the Is_Sliced attribute of the work effectively propagates the *sliced* condition of the TT task until the terminal slot, for which Is_Continuation will be False.

The scheduler must also perform some actions upon calls to its public services. These may affect to the work status and to the priority of the underlying threads. It is the task itself who changes its own work status and priority, if needed, while running a scheduler protected

Table 1: Some actions to be taken by the scheduler at a slot switch. Actions with regard to the exhausted slot (*Actions at END of slot*) are shown in the top part, Actions related to the immediately starting slot (*Actions at START of slot*) take the bottom part. These actions depend on work status and type of slot.

Work status		**Actions at END of slot**
`Has_Completed`	`Is_Sliced`	
False	True	Hold task
False	False	Raise `Program_Error`

Work status			Next Slot	**Actions at START of slot**
`Has_Completed`	`Is_Sliced`	`Is_Waiting`	`Is_Optional`	
True	True			Common Actions (CA)*
True	False	True		Release Task + CA*
True	False	False	True	CA*
True	False	False	False	Raise `Program_Error`
False	True			Resume Task + CA*

*Common Actions ≡ Work.Is_Sliced ← Slot.Is_Continuation; Set_Handler

Table 2: How and when the work status and TT task priority are modified by the scheduler.

Invoked procedure	**Changes to work status**			**Changes to task prio**
	`Has_Completed`	`Is_Sliced`	`Is_Waiting`	
`Wait_For_Activation`	True		True	`Priority'Last`
`Continue_Sliced`		True		
`Leave_TT_Level`	True			`Task'Base_Priority`

operation. Table 2 summarises these update operations. For example, when a TT task invokes Leave_TT_Level, its work status is marked as completed and its priority demoted to the task's base priority – so the base priority of the task implementing the TT pattern must be determined according to the required priority level when it runs in the PB level. For readers interested in the full code of the TT scheduler, it is available on GitHub [8].

5 Experimental Results

To evaluate the performance of the scheduler, mainly in terms of jitter, we have used a workload similar to the one we used in [7]. In this case, the hardware platform is a STM32F4 Discovery board at 168 MHz clock frequency, running a modified version of the GNAT *ravenscar-full* runtime from AdaCore's GNAT GPL 2017.

Apart from adding the Ada.Dispatching.TTS package to the runtime, we have changed the timing event and delay resolution of the GNAT runtime from the original $1\ ms$ to $10\ \mu s$. We have measured an additional overhead of 3.5% due to this modification.

Figure 9 shows the cumulative frequency histogram of measured release jitters of three TT tasks (`W1`..`W3`), one simple and two I-F tasks, and two PB tasks (`T4` and `T5`). The jitter is measured with respect to the theoretical activation time, i.e. the start of the next slot for TT tasks or the time expression used in the delay statements of the periodic PB tasks. As it happened with the previous full-Ada implementation, the PB tasks suffer from quite disperse release jitter, due to interference from TT tasks and higher priority PB tasks. Their minimal jitter is however shorter than that of the TT tasks, due to the TT scheduler overhead. Note that the jitter for TT tasks is not only short but also very predictable, always within the range of 22 to 24 μs.

Taking this high predictability into account, we have tested an optimisation strategy that requires precise characterisation of the TT scheduler overhead. The optimisation consists in reprogramming the TT scheduler for the next slot time minus a fixed offset (20 μs in our case) so that the TT scheduler overhead is not paid at the start, but at the end of the TT slot. This overhead is unavoidable and has to be taken into account when the plan is built, but moving it to the end of the slot drastically improves the *release jitter* of the TT tasks. Figure 10 shows the release jitters obtained after applying this optimisation, which now range from 3 to 4 μs. These results clearly outperform those previously obtained with a full-Ada implementation using a MarteOS [1] on a Pentium @ 800 MHz, that raged from 30 to 70 μs.

6 Conclusions

This paper has presented the results of transforming a full-Ada architecture for combined TT-PB scheduling [7, 6] to make it Ravenscar-compatible. Our aim was to make this scheduling strategy compatible with a more appropriate programming model for High-Integrity systems. Our first efforts focused on a user-level library supporting the scheduler, but we soon moved to a runtime library, so that the scheduler could support continuation slots (via hold/resume) and non-TT slots (via priority demotion), thus improving expressiveness and making room for more possible patterns. We have also introduced the concept of optional slot, and included provision for, now tolerable, no-show situations.

In addition, we have made Ada.Dispatching.TTS a generic package, where the number of regular work identifiers is a generic parameter. This allows us to keep the size of data structures to the minimum necessary for the number of TT tasks to be scheduled. The experimental results are encouraging, even better than those obtained in full-Ada with a much faster processor. No doubt, the simplicity of the Ravenscar runtime has to do with these results.

References

[1] M. Aldea and M. González-Harbour (2001), *MaRTE OS: An Ada Kernel for Real-Time Embedded Applications*, Reliable

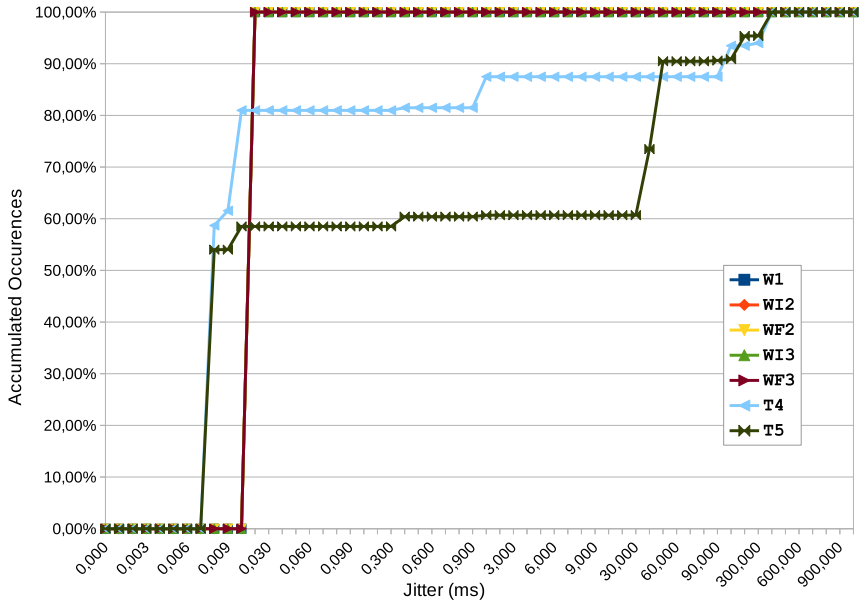


Figure 9: Release jitter on a STM32F4 Discovery at 168 MHz

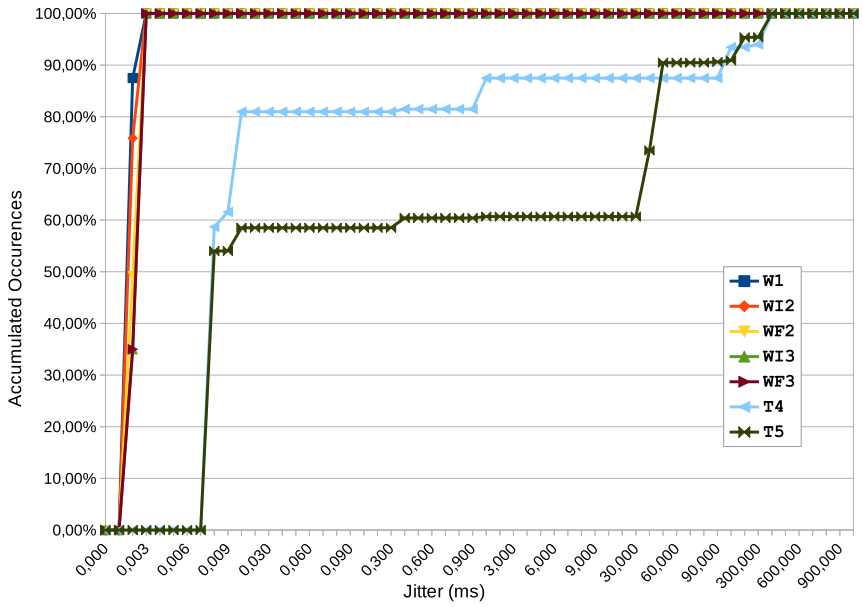


Figure 10: Release jitter with overhead anticipation

Software Technologies - Ada Europe 2001, Lecture Notesa in Computer Science, 2043:305–316.

[2] ISO/IEC-JTC1-SC22-WG9 (2012), *Ada Reference Manual ISO/IEC 8652:2012(E)*, `http://www.ada-europe.org/manuals/LRM-2012.pdf`.

[3] J. Leung and J. Whitehead (1982), *On the complexity of fixed-priority scheduling of periodic, real-time tasks*, Performance Evaluation (Netherlands), 2(4):237–250.

[4] C. Liu and J. Layland (1973), *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*, Journal of the ACM, 20(1):46–61.

[5] J. Real and P. Rogers (2016), *Session Summary: Experience*, Ada Letters, 36(1):101–102.

[6] J. Real, S. Sáez, and A. Crespo (2016), *Combined scheduling of time-triggeed plans and priority scheduled task sets*, Ada Letters, 36(1):68–76.

[7] J. Real, S. Sáez, and A. Crespo (2016), *Combining time-triggered plans with priority scheduled task sets*, In M. Bertogna and L. M. Pinho, editors, Reliable Software Technologies – Ada-Europe 2016, volume 9695 of Lecture Notes in Computer Sciencem, Springer.

[8] S. Sáez and J. Real (2018), *TTS Ravenscar runtime*, https://doi.org/10.5281/zenodo.1168723.

Position Paper: Clock Support in Ada

Kristoffer Nyborg Gregertsen
SINTEF Digital, Norway; email: kristoffer.gregertsen@sintef.no

Abstract

This position paper briefly revises the clock support in the Ada programming language, including execution time clocks and timers for tasks and interrupts. It is argued that the standard library would benefit from a more coherent handling of clocks, in particular with in the lack of a shared interface for clocks and timers, and that better support for high-precision real time (TAI/UTC) should be provided.

1 Background

Most real-time control systems are of a distributed nature and need to have a shared system time of sufficient precision between nodes. Examples of such systems can be found within aerospace, robotics and automation domains. Having a shared system time allows measurements from different nodes to be correctly time-stamped and correlated, and synchronous events to be orchestrated. As all clocks have a certain drift due to inaccuracies in the oscillators and temperature changes, it is necessary to *discipline* them by distributing a time signal to compensate for this drift.

The passing of time is defined by International Atomic Time (TAI) reference, that is made up of a number of atomic clocks around the world distributing time signals between them. The measurements from these clocks are compensated for that time passes at slightly different pace due to differences in gravity. The TAI time reference is *monotonic* – it does not have any jumps. UTC time is based on TAI, but is compensated for the variances in earths rotation by adding an occasional leap seconds at the end of the year. Leap seconds are announced in advance, and special care needs to be taken to handle them properly in systems that use UTC time. As of May 2018, TAI is 37 seconds ahead of UTC due to leap seconds.

For most purposes the Network Time Protocol is the most convenient way of getting access to UTC. This protocol continuously retrieves time from time servers and estimates the drift of the computers internal clock, continuously adjusting it so to avoid abrupt changes. The precision of NTP is in the millisecond range, but can be subjected to offset and jitter if the latency of the network is not consistent. The Precision Time Protocol (PTP) achieves sub-microsecond precision by measuring the latency through the network, but also requires special Ethernet switches and network cards. PTP is used to keep distributed computer systems time synchronized, for instance to acquire scientific measurements. For instance, the White Rabbit Project [4] at CERN uses PTP and special hardware to synchronize experiments over a large area.

To get access to a high-precision UTC signal that can be distributed in the local network it is most convenient to use a GNSS receiver, listening to GPS/Galileo/GLOSNASS/BeiDou satellites that distribute UTC time. Such clocks will typically be synchronized with UTC at sub-microsecond offset, and distribute time signals with protocols such as PTP or NTP. These protocols are supported by operating systems such as GNU/Linux, Mac OSX, Windows, QNX and others. It is also possible to add support in embedded systems such as the Xilinx UltraScale+ MPSoC by using PTP IP cores, or by using a combination of time messages and Pulse Per Second (PPS) signals in embedded systems to correct the internal oscillator.

2 UTC support in Ada

The package Ada.Calendar is the standard way of getting the system time in Ada, and has been present since Ada 83. The language defines the type Duration for time duration, and the type Time for time instants that can be retrieved by the Clock function, and functions for getting the year, month, day and intra-day second of that time instants. The standard requires that the value of Duration'Small should be no greater than 20 microseconds, but it could also be of nanosecond precision if the implementation allows. The constant System.Tick gives the duration of which the clock will return the same value. Subpackages of Ada.Calendar support time-zones, retrieval of UTC offset, and time arithmetic including historical leap seconds.

For use in real-time systems, annex D of the standard has had support for high-precision monotonic time since Ada 95. This package has more stringent requirements for precision, and also requires the clock to be monotonic with no abrupt changes as this would lead to problems for real-time applications, for instance when using periodic tasks with **delay until**. The package supports getting the unit for both time instants and time spans, and the standard requires that these shall be no greater than 20 microseconds. Also, it is required that the tick of the real-time clock shall be no greater than 1 milliseconds. Implementations are required to document the metrics of the clock compared to the TAI reference, such as drift and maximal clock jumps. As stated before, a clock that is not disciplined by a time signal will drift over time, and this drift rate will vary among individual clocks and with temperature. It will thus be hard for implementations to give tight bounds on the drift for undisciplined clocks, and for disciplined clocks it would be more relevant to give a bound to the maximal offset to UTC and information of how time is adjusted. For

general purpose operating systems like GNU/Linux, it would of course be hard to document this at all as it depends on how the system administrator has configured the system.

The developer should be able to know whether a reliable clock synchronization is available or not. This is must be done as a query at run-time, for instance using a routine as Last_UTC_Synchronization to get the time point at which the clock was last synchronized, or a function for querying the a more general state of the clock synchronization. It does not suffice to know that the the system *supports* time synchronization, as this synchronization can be broken due to a range of different errors that can not be predicted in advance, such as network error or snow on the GNSS antenna. Also, it may take some time at start-up to achieve time synchronization.

Furthermore, even though the real-time clock may be implemented by a system clock that is kept in sync with TAI, there is no standard way of retrieving an UTC timestamp from the real-time clock. Such a timestamp could be made from the calendar package, but even though the standard advises that the two clocks could be transformations of each other, there does not seem to be a straightforward way of doing this transform. Also, the calendar package does not take leap seconds into account, something that is needed for correct UTC timestamps. Another problem for real-time applications is that the much used Ravenscar profile disallows the use of the calendar package, leaving no standard method of getting absolute time.

It is argued that means should be provided for real-time applications to acquire high-precision UTC timestamps, given that the implementations supports a clock that is disciplined relative to TAI, and that the metric of this time synchronization should be documented. Specifically, a set of functions should be provided to translate between UTC timestamps and the Time type of Annex D. Such a feature would be of great value to distributed systems that need to communicate correlated measurements, and execute actions at specific points time.

3 Coherent clocks in Ada

Related to real-time clocks and timers, there should be a discussion of how to make these more coherently represented in the language. In addition to the standard calendar and real-time clock described earlier, Ada 2005 defined timing events for the real-time clock, clocks and timers for the execution time of tasks, and group execution time budgets for tasks. Also, Ada 2012 addressed concerns about the accuracy of execution time measurement for tasks, by adding support for measuring the execution time of interrupts, both for individual interrupt ID's and the total execution time for interrupt handling. However, full execution time control for interrupts is not possible as interrupt execution time timers and group budgets are not supported. At IRTAW-16 the author proposed to add execution time timers for interrupt handling as a new type extending the existing task timers [3]. This non-standard feature was earlier implemented on GNATforAVR32 [1, 2]. However, IRTAW-16 found that the proposal needed further work, and that a more coherent type system with a new root type should be specified [5]. At, IRTAW-18 the author proposed a revised class hierarchy for timers and group budgets to support interrupts in a more coherent way, and showed how the deferrable server for interrupt handling could be implemented with this feature. This proposal was rejected on breaking backward compatibility by adding a root class for Timer.

In the authors opinion, the different clocks and timers in Ada are defined in a fragmented way, that does not allow for reuse and abstraction through object-orientation. The different clocks packages are defined in a similar way, but have no common interface, and even though the timers and timing events are quite similar and are defined as tagged types, there is no common base class or shared interface that allow them to be used in an object-oriented manner. It would seem that the only real benefit of using tagged types for these types is that it allows for dot notation on routine calls. By redesigning the clock definitions using common interfaces, it would be possible to have reuse of timing related functionality such as time servers, and also to define bespoke clocks for real-time systems if needed. Such as object-oriented design is followed for the Real-Time Specification for Java (RTSJ) that also allows custom clocks [6]. The new package <chrono> of C++11 also supports different clocks as defined types, and the coming C++20 will also support TAI, GPS and other clock sources. As of now RTSJ and C++ seem ahead of Ada in terms of advanced and coherent clock support, and it is argued that the value of backward compatibility should be weighted against the need to keep Ada at the forefront within real-time systems.

References

[1] K. N. Gregertsen and A. Skavhaug (2010), *Implementing the new Ada 2005 timing event and execution time control features on the AVR32 architecture*, Journal of Systems Architecture, 56(10):509–522.

[2] K. N. Gregertsen and A. Skavhaug (2011), *Implementation and Usage of the new Ada 2012 Execution Time Control Features*,Ada User Journal, 32(4):265–275.

[3] K. N. Gregertsen and A. Skavhaug (2013), *Execution time timers for interrupt handling*, ACM SIGAda Ada Letters, 33(2):87–96.

[4] J. Serrano, M. Lipinski, T. Wlostowski, E. Gousiou, E. van der Bij, M. Cattin, and G. Daniluk (2013), *The White Rabbit project*, Proceedings of IBIC2013.

[5] T. Vardanega and R. White (2013), *Session summary*, ACM SIGAda Ada Letters, 33(2):126–130.

[6] A. Wellings and M. Schoeberl (2011), *User-Defined Clocks in the Real-Time Specification for Java*, Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems - JTRES '11, page 74.

Ada User Journal

Call for Contributions

Topics: **Ada, Programming Languages**, **Software Engineering Issues** and **Reliable Software Technologies** in general.

Contributions: **Refereed Original Articles**, **Invited Papers**, **Proceedings** of workshops and panels and **News and Information** on Ada and reliable software technologies.

More information available on the
Journal web page at

http://www.ada-europe.org/**auj**

Online archive of past issues at http://www.ada-europe.org/auj/archive/

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o KU Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Hubert B. Keller
Karlsruher Institut für Technologie (KIT)
Institut für Angewandte Informatik (IAI)
Campus Nord, Gebäude 445, Raum 243
Postfach 3640
76021 Karlsruhe
Germany
Email: Hubert.Keller@kit.edu
URL: ada-deutschland.de

Ada-France

attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada-Switzerland

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch