The journal for the international Ada community

Ada User Journal

Volume 43
Number 2
June 2022

Produced by Ada-Europe

Information on Subscriptions and Advertisements

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

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

ADA USER JOURNAL

Volume 43
Number 2
June 2022

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

I would like to start this editorial by celebrating the fact that the Ada-Europe International Conference on Reliable Software Technologies (AEiC-2022) is taking place once again as a physical event, this time in Ghent, Belgium, on June 14-17. The conference features an excellent and diversified technical program, also including a rich set of opportunities for socializing, namely by participating in very interesting cultural and social events.

Coincidentally, but not surprisingly, in this issue of the Ada User Journal we start the publication of the AEiC-2022 Work-in-Progress Session proceedings. We include eight contributions, covering a set of quite diverse topics related to the development of reliable embedded systems, or to the Ada language.

The first two papers address Ada-related features and are both written by authors from AdaCore (USA and France). They report on work in progress concerning the definition of a pattern-matching feature for Ada, and the definition of a work-stealing scheduler for the lightweight threads supporting the Ada 2022 parallel programming features. Then, a paper describing a new scheduling method for mixed-criticality systems is presented. The work, by authors from the University of York, UK, is developed for multi-core systems and considers the survivability of low criticality tasks, to achieve consistent schedules for different operation modes. After that, two papers by authors from the Faculty of Sciences of the University of Lisbon are included. In both cases, machine learning techniques are used for improving aspects of embedded systems operation, either the quality of sensor data or the communication reliability. The following paper is authored by Marc Solé and Leonidas Kosmidis, who are affiliated with the Barcelona Supercomputing Center (BSC) and the Polytechnic University of Catalonia (UPC). They report on their experience using the GCC and LLVM compilers, which were extended with the necessary support for the SPARROW hardware, an AI-oriented SIMD Extension. Then we include another paper authored by BSC/UPC researchers, but in this case co-authored by David Steenari, from the European Space Agency. The paper presents ongoing work concerning the acceleration of data compression, considering two different embedded GPU platforms for space systems for evaluation purposes. It is shown that despite the typically sequential nature of data compression tasks, it is possible to achieve performance improvements through parallelization. Finally, this first part of the AEiC-2022 WiP proceedings is closed with a paper authored by Zineb Boukili, Hai Nam Tran and Alain Plantec, from the University of Brest. The paper presents a new approach for run-time timing monitoring of real-time systems, which is based on the definition of fine-grained program blocks and on code instrumentation on defined monitoring points.

In this AUJ issue we also include, as usual, the News Digest section prepared by Alejandro R. Mosteo, and the Calendar and Events sections, prepared by Dirk Craeynest.

Antonio Casimiro
Lisboa
June 2022
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Alejandro R. Mosteo

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

Contents

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

Preface by the News Editor

Dear Reader,

It is a recurrent topic that the several GNAT versions with differing licensing conditions are [not] a hindrance to Ada adoption. Well, AdaCore has announced that, moving forward, their open source oriented compiler offering will be unique and based on the FSF source tree, hence with similar licensing as other GCC languages [1]. Let us hope this puts to rest the FUD of the past. No doubt, a remarkable piece of news and a valiant bet on the future of Ada with the backing of a healthy open source community.

Likewise on the open source front, the HAC compiler continues making steady progress [2], with several versions released in close succession, each one bringing more Ada features into its supported subset.

This issue has seen a livelier than usual section on Ada and other languages. I always find interesting the examination of the interrelations between languages, and how Ada fits in the past, present and future of programming languages. Will Ada ever regain a spot in the top-10 more popular languages [3]? Perhaps we should devise our own ranking using Metrics that Really Matter™ ;-)

Sincerely,
Alejandro R. Mosteo.

[1] "A New Era for Ada/SPARK Open Source Community", in Ada-related Tools.

[2] "HAC v.0.2", in Ada-related Tools.

[3] "When Ada Was the Most Popular Language", in Ada and Other Languages.

Ada-related Events

Ada Monthly Meeting Proposal

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Subject: Ada Monthly Meeting proposal
Date: Tue, 26 Apr 2022 21:59:32 +0200
Newsgroups: comp.lang.ada

Ada Monthly meeting

A lot of programming languages and libraries have meetings/meetups which allow the community to come together and have a chat, share ideas, proposals and better utilise and prioritise resources. I would like to propose such a thing for Ada. Below is the rationale and some ideas and issues.

Motivation

The Ada community does not have many members when compared to other more-well-known communities. However, there is still some interest in having such type of meetings. This was recently made clear after some people pointed out they would like to have such a thing in the Ada channel over at Gitter. Personally, I have been playing with such an idea and that is what motivated me to volunteer to drive the FOSDEM Ada devroom. I would like the Ada community to be more well-known and to have the same "resources" as other communities have. Meetups are a great way to have fun while discussing what we like.

In meetups, generally speaking, users and developers can have the opportunity to come together, discuss topics, organise resources and help each other. Gray-beards can help those who still have colour in their hair; people with different sets of skills can propose solutions to problems that one may not have thought about; authors can present their work or improvements, etc.

I would like to use the Fortran community as an example of what meetups can be used for. Here is their April monthly-meetup [1]. Their meetups are very focused on the language and "core" tooling... it is quite formal, which may not be what I had in mind, but we will see.

Who is this meant for?

Everybody who is interested.

I would love to see some participation of the "Industrial users". But I understand that a lot of people see Ada (and many other things) as a tool that brings food to the table, nothing more. So I would not expect much participation from this group.

Newbees and beginners are also more than welcome. They could see what people are doing and ask questions that are better answered in real-time by a person, instead of a Stackoverflow for example.

Though, I must be honest, it is mostly intended for people who are interested in the Ada environment and open side of things. This is due to the nature of an open discussion and building a community. I have to be clear and state that I am biased towards the libre community, so feel free to point out any unfairness.

How would it work? What would it be like?

THIS IS JUST A PROPOSAL, SO TAKE THIS AS SUCH.

I thought about having a Jitsi room (libre conference system that runs on your browser, same one used in FOSDEM) [2] where people can just join and take part of the meetup. Jitsi allows for moderation too, so that speakers can talk without getting interrupted and it has a built in chat too.

So, what could be discussed? Here is a short list of ideas that I have:

- Monthly news: new releases, milestones, etc.
- Presentations: attendees may want to present their work or do a demonstration. They may also want to have a discussion about a specific topic (for example, the use of Ada 2022 features).

- General libre software coordination: improvements to tools, feedback, questions, past goals discussion, etc.
- General Q&A related to Ada and open to everybody.
- Finally, a beer.

I think this could take place between 30 min to 2 hours, depending on the load of that day. Presentations would obviously be much more casual and easy when compared to an actual conference.

Potential issues

1. Not enough interest.
2. Timezones! Users are mostly concentrated in Asia-Pacific/EU/USA, which makes coordination an absolute pain. A compromise could be found, or a different schedule each month in such a way that everybody benefits (and gets screwed) equally.
3. Organisation: there needs to be a main organiser and a second in command-
4. There also needs to be a medium in which to spread the word. C.L.A is a good starting point, but may not reach the wider community. It could be announced everywhere every month, but that is a tedious task.

Feedback

I have probably said enough, even if not everything has been said. So I would like to ask for your feedback and specially know if you would be interested.

Thank you for your time,
Fer

References

[1] https://invidious-us.kavin.rocks/watch?v=8-_ll4f0gN8

[2] https://meet.jit.si/

From: Maxim Reznik <reznikmm@gmail.com>
Date: Wed, 27 Apr 2022 03:34:13 -0700

I would give it a try!

From: Anton F. <imantonmeep@gmail.com>
Date: Wed, 27 Apr 2022 05:57:49 -0700

I would participate, this is a great idea!

From: Yossep Binyoum <yossep237@gmail.com>
Date: Thu, 28 Apr 2022 13:41:03 -0700

From Senegal, I totally agree with you. I give it a try

From: Stéphane Rivière <stef@genesix.org>
Date: Fri, 29 Apr 2022 16:38:39 +0200

Great idea!

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 29 Apr 2022 11:29:47 -0700

> * Ada Monthly meeting

Sounds interesting. I maintain Emacs Ada mode; this might be a good forum to get less formal feedback than the ada-mode mailing list provides, and to hear what other IDEs are doing for Ada.

> 1. There also needs to be a medium in which to spread the word. C.L.A is a good starting point, but may not reach the wider community. It could be announced everywhere every month, but that is a tedious task.

This sounds like a job for a bot; post the same announcement to a list of channels.

Anyone have a bot written in Ada?

From: Maxim Reznik <reznikmm@gmail.com>
Date: Fri, 29 Apr 2022 21:04:15 -0700

> Anyone have a bot written in Ada?

I have a bot in Ada for Telegram. It is a bridge between Telegram, Jabber, IRC channel. It also checks whether telegram newcomers are not bots.

I can write another one for announcements, but I'm not sure if announcing once a month is worth the time :)

From: Ada Forge <adaforge2022@gmail.com>
Date: Sat, 30 Apr 2022 06:57:40 -0700

> * Ada Monthly meeting

Nice initiative!

Take me into account ;-)

Some subjects I'd love to debate with connoisseurs:

* UTF8-Unicode-UCS: a lot of libraries are offering strings manipulation. State of the art? (Gnat extensions, GnatColl, Matreska, Gnoga, ...)
* OS system usage (as (system shell) scripts, in place of Perl, Python, …): GNAT extensions; Florist; GnatColl; SoWebIO; ...
* Windowing (2D) systems: future of GTK/Glade; Qt6/Qt Design Studio; GWindows; Apple new SwiftUI MV paradigm; wxWidgets; Tk/TCL
* How let anyone collaborate to AdaForge's new up-todate 2022 Ada resources gathered all over the internet ;-) Through GitHub?

Cheers, with a fresh Belgian Ada 10°
William

From: Rod Kay <rodakay5@gmail.com>
Date: Sun, 1 May 2022 02:33:47 +1000

> Nice initiative!

Agree. Count me in Fer :).

> Some subjects I'd love to debate with connoisseurs:

If by 'OS system usage' you mean using Ada to write shell-like scripts then you may be interested in aShell. It builds on Florist to allow Ada applets to more easily call and interact with OS commands.

The last release allowed OS commands to be called but only from a single Ada task. Atm, work is being done on supporting task safe commands (via a spawn manager).

The next release will contain the task safe commands and be Alire enabled, and should occur in the next month or so (Lady Ada willing).

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 30 Apr 2022 18:01:24 +0100

> Some subjects I'd love to debate with connoisseurs:

> * UTF8-Unicode-UCS: a lot of libraries are offering strings manipulation. State of the art? (Gnat extensions, GnatColl, Matreska, Gnoga, ...)

> * Windowing (2D) systems: future of GTK/Glade; Qt6/Qt Design Studio; GWindows; Apple new SwiftUI MV paradigm; wxWidgets; Tk/TCL

Oh, I suppose I'll have to attend given I have experience with those.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Tue, 3 May 2022 21:06:21 +0200

Thank you all for your answers :)

It seems that there is some interest to have a meeting from time to time. Other communication channels where this proposal was posted did have other people who liked the idea. For this reason, I would like to share some extra bits.

- I think the duration of such a monthly meeting could last for an hour, an hour and a half.
- Could take place monthly in a varying schedule to suite some people better than other depending on the month.
- The level of preparation is much much lower than FOSDEM or similar venues. This is more about building community than actually making this a serious thing.
- The structure could be the following:
  - < 5 mins to share important news and announcements.
  - < 30 mins reserved for already predefined talks/topics. More on this later.
  - < 15 min discussion topic. Maybe there is something that needs a few words, it's the topic of the day.
  - < 15 mins to let people share their work or improvements.
  - The main meeting ends here.

- Open questions and answers and general discussion/beer.
- The main section of the meeting (without Q&A and open discussion) could be recorded and uploaded to video hosting sites. I know a few people already reupload the videos from FOSDEM, so I could as them to do the same for us. This would allow us to keep a log of the meetings:)
- We could use Jitsi, a libre conference software. I know it has recording capabilities, but I think only for Youtube... :/ We will see whether Jitsi actually works or not...

If this works, I think we could start inviting people to share their work and reserve time for their presentations. That is what the second section of the proposed schedule is about. At the beginning, obviously, we will focus on making sure that the meetings work and see if there is enough recurring interest in them.

Regarding the actual planning. I will not make it for the month of May unless someone steps and helps a fair bit. On a personal note, I have a lot of work and it will just keep increasing so I cannot ensure that I will be able to pull something like this alone. FOSDEM was already a bit exhausting :P

I also want to see what you have to offer both in direct help or if you have projects that you want to talk about, presentations, etc. Some of you already commented on it, so I am happy.

What is your opinion about this? I would need feedback :)

Also, please, feel free to repost this to other social media. The more Ada users and people interested in Ada the better! If you want a contact, feel free to email me at "irvise(AT)irvise.xyz".

Bye now :D
Fer

CFP: ACM SIGAda HILT 2022 Workshop at ASE '22, October 14, 2022

From: Tucker Taft <tucker.taft@gmail.com>
Subject: CFP: ACM SIGAda HILT 2022 Workshop at ASE '22, October 14, 2022
Date: Thu, 12 May 2022 18:30:15 -0700
Newsgroups: comp.lang.ada

Please consider contributing to this workshop sponsored by ACM SIGAda: HILT-2022 - Supporting a Rigorous Approach to Software Development

This is the seventh in the HILT series of conferences and workshops focused on the use of High Integrity Language Technology to address challenging issues in the engineering of highly complex critical software systems.

High Integrity Language Technologies have been tackling the challenges of building efficient, safe, reliable software for decades. Critical software as a domain is quickly expanding beyond embedded real-time control applications to the increasing reliance on complex software for the basic functioning of businesses, governments, and society in general.

For its 2022 edition, HILT will be a workshop of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE'2022. The workshop will be held on October 14th 2022.

See ASE'2022 (https://conf.researchr.org/home/ase-2022) for details on the venue and registration.

Topics

HILT 2022 will focus on the increasing synergies between formal methods (theorem provers, SAT, SMT, etc.), advanced static analysis (model checking, abstract interpretation), software design and modeling, and safety-oriented languages. From separate fields of research, we now observe a stronger interconnection between formal methods, advanced analytics, modeling and design of software, and safety features in programming languages. Programming languages for safety-critical systems now routinely integrate theorem proving capabilities like C/ACSL or Ada/SPARK2014. Theorem provers such as Coq, Lean, or Isabelle have established themselves as a viable strategy to implement compilers or properly define the semantics of domain-specific languages. Tools for verifying modeling languages such as AADL, Lustre, and Simulink are becoming more widely available, and with the emergence of the Rust language and the release of Ada 2022, safety is rising to the top of concerns for critical systems developers.

The HILT'2022 workshop seeks to explore ways High Integrity Language Technologies leverage recent advances in practical formal methods and language design to deliver the next generation of safety-critical systems.

Call for Papers

This workshop is focused on the practical use of High Integrity languages, technologies, and methodologies that enable expedited design and development of software-intensive systems.

Key areas of interest include experience and research into:

Practical use of formal methods at industrial scale

IDE-support for formal methods

Model-level analysis tools for systems like SysML, AADL, Lustre, or Simulink

Continuous Integration and Deployment based on advanced static analysis tools

Safety-Oriented Programming Language features *Qualification of Language Tools for critical systems use

The workshop accepts either short abstracts (2-3 pages) for presentation, or full papers (up to 8 pages).

Submissions should conform, at time of submission, to the ACM Proceedings Template: https://www.acm.org/publications/proceedings-template.

The workshop proceedings will be published in the ACM Ada Letters. Authors of accepted papers will be invited to contribute to a special issue of the Springer Journal on Software and Tools for Technology Transfer (STTT).

Paper submission

Submit your paper through Easychair at https://easychair.org/conferences/?conf=hilt22

Important Dates

Submission Deadline: July, 1 2022

Notification to authors: August, 1 2022

Workshop Date: October 14th 2022.

From: Tucker Taft <tucker.taft@gmail.com>
Date: Thu, 12 May 2022 18:34:51 -0700

> Please consider contributing to this workshop sponsored by ACM SIGAda: HILT-2022 - Supporting a Rigorous Approach to Software Development

Website is:
https://conf.researchr.org/home/hilt-2022

Press Release - AEiC 2022, Ada-Europe Reliable Softw. Technol.

[The event took place during 14-17 June, so this announcement is for the record. —arm]

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Press Release - AEiC 2022, Ada-Europe Reliable Softw. Technol.
Date: Sun, 12 Jun 2022 20:59:37 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

FINAL Call for Participation

***UPDATED Program Summary ***

26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022)

14-17 June 2022, Ghent, Belgium

www.ada-europe.org/conference2022

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

#AEiC2022 #AdaEurope #AdaProgramming

*** Final Program available on the conference web site ***

*** Add tutorials and/or a workshop to your conference registration ***

www.ada-europe.org/conference2022/tutorials.html

* Welcome Event on Tuesday evening *

Press release:

26th Ada-Europe Int'l Conference on Reliable Software Technologies.

International experts meet in Ghent.

Ghent, Belgium (12 June 2022) - Ada-Europe together with the University of Ghent, Belgium, organizes from 14 to 17 June 2022 the 26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022). The event is in cooperation with the Ada Resource Association (ARA), and with ACM's Special Interest Groups on Ada (SIGAda), on Embedded Systems (SIGBED) and on Programming Languages (SIGPLAN).

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

This year's conference offers 4 tutorials, a keynote and an invited presentation, a technical program of 7 sessions with refereed papers industrial, work-in-progress, and vendor presentations, a social program with exciting sightseeing, 2 workshops and a Birds-of-a-Feather session.

Four tutorials are scheduled on Tuesday, targeting different audiences:

- "Moving up to Ada 2022", by S. Tucker Taft, AdaCore, USA;
- "Numerics for the non-numerical analyst", by Jean-Pierre Rosen, Adalog, France;
- "The ALiRe Package Manager", by Fabien Chouteau, France, and Alejandro Mosteo, Spain;
- "The HAC Ada Compiler", by Gautier de Montmollin, Switzerland.

Vendors and organisations will be present in the networking area on Wednesday and Thursday include AdaCore, VECTOR, and Ada-Europe.

Two eminent speakers have been invited to deliver a talk on each of the core conference days:

- on Wed Jun 15, a spotlight talk (remote) by Anita Carleton, Software Engineering Institute, Carnegie Mellon University, USA, about "Envisioning the Future of Software Engineering";
- on Thu June 16, a keynote talk by Cristina (Crista) Lopes, School of Computer Sciences, University of California at Irvine, USA, who will present her study on "The Curious Case of Code Duplication in Github".

The technical program on Wednesday and Thursday presents 7 sessions with 9 journal-track refereed technical papers, 9 industrial, 12 work-in-progress, and 2 vendor presentations in sessions on: Uses of Ada, Real-Time Systems 1, Development Challenges, Advanced Systems, Special-Purpose Systems, Verification Challenges, Real-Time Systems 2.

On Friday the conference hosts for the 7th year the workshop on "Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering" (DeCPS 2022), as well as the International Workshop "AADL Unveiled by its Practitioners (ADEPT), and a Birds-of-a-feather (BoF) Meeting on the "Future of ASIS and Vendor Independent Tools".

Peer-reviewed papers have been submitted to a special issue of the Journal of Systems Architecture and are heading towards final acceptance as open-access publications. Industrial and work-in-progress presentations, together with tutorial abstracts, will be offered publication in the Ada User Journal, the quarterly magazine of Ada-Europe.

The social program includes for all tutorial and conference participants on Tuesday evening a Welcome Aperitif with beer tasting (sponsored by VECTOR) in the "Il Trovatore" lounge, a restored medieval cellar. On Wednesday evening, a private visit to the Gothic-style St Bavo's Cathedral and its artistic treasures including the world-famous Lam Gods altarpiece, followed by the Conference Banquet in the Abt, the only brasserie from the famous Orval Trappist beer brewery. And on Thursday evening a boat tour in the canals that encircle the medieval center of Ghent, followed by a conference dinner at the Carlos Quinto restaurant, a short walk across the heart of town from the boat pier.

The Best Presentation Award will be offered during the Closing session.

The full program is available on the conference web site.

Online registration is still possible.

Latest updates:

The 16-page "Final Program" is available at www.ada-europe.org/conference2022/docs/AEiC_2022_Final_Program.pdf.

Check out the tutorials in the PDF program, or in the schedule at www.ada-europe.org/conference2022/tutorials.html.

Registration is done on-line. For all details, select "Registration" at www.ada-europe.org/conference2022 or go directly to https://registration.ada-europe.org.

A printed Conference Booklet with abstracts of all technical papers

and industrial presentations will be included in every conference

handout, and is available at www.ada-europe.org/conference2022/docs/AEiC_2022_Booklet_of_Presentations.pdf.

AEiC 2022 is sponsored by Ada-Europe (www.ada-europe.org), AdaCore (www.adacore.com), and VECTOR (www.vector.com/int/en/products/products-a-z/software/vectorcast).

Help promote the conference by advertising it.

Recommended Twitter hashtags: #AEiC2022 #AdaEurope #AdaProgramming.

Our apologies if you receive multiple copies of this announcement.

Please circulate widely.

Dirk Craeynest, AEiC 2022 Publicity Chair

Dirk.Craeynest@cs.kuleuven.be

* 26th Ada-Europe Int. Conf. Reliable Software Technologies (AEiC 2022)

* June 14-17, 2022, Ghent, Belgium * www.ada-europe.org/conference2022

(V6.1)

Ada/SPARK Crate of the Year 2022

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Subject: Ada/SPARK Crate Of The Year is back!
Date: Tue, 28 Jun 2022 05:55:20 -0700
Newsgroups: comp.lang.ada

https://blog.adacore.com/announcing-the-2022-ada-spark-crate-of-the-year-award

[AdaCore offers 3 prizes of $2,000 each for the following categories: best overall

Ada crate, best crate written in SPARK and/or contributing to the SPARK ecosystem, and best Ada or SPARK crate for embedded software. Candidates can be submitted until the end of the year. —arm]

Ada-related Resources

[Delta counts are from May 9th to July 18th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: 18 Jul 2022 14:53 CET
To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn: 3_328 (+26) members [1]
- Reddit: 8_078 (+73) members [2]
- Stack Overflow: 2_238 (+26) questions [3]
- Libera.Chat: 75 (=) concurrent users [4]
- Gitter: 123 (+8) people [5]
- Telegram: 143 (+4) users [6]
- Twitter: 30 (=) tweeters [7]
  75 (+22) unique tweets [7]

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

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

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

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

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

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

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

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: 18 Jul 2021 15:16 CET
To: Ada User Journal readership

Rosetta Code: 915 (+15) examples [1]
  39 (+1) developers [2]
GitHub: 763* (=) developers [3]
Sourceforge: 244 (-30) projects [4]
Open Hub: 214 (=) projects [5]
Alire: 260 (+17) crates [6]
Bitbucket: 87 (-1) repositories [7]
Codelabs: 53 (=) repositories [8]
AdaForge: 8 (=) repositories [9]

*This number is unreliable due to GitHub search limitations.

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

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

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

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

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

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

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

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

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

Language Popularity Rankings

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada in language popularity rankings
Date: 18 Jul 2021 15:51 +0100
To: Ada User Journal readership

[Positive ranking changes mean to go up in the ranking. —arm]

- TIOBE Index: 30 (-3) 0.38% (-0.08%) [1]
- PYPL Index: 17 (=) 0.86% (+0.05%) [2]
- IEEE Spectrum (general): 31 (=) Score: 38.8 (=) [3]
- IEEE Spectrum (embedded): 9 (=) Score: 38.8 (=) [3]

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

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

[3] https://spectrum.ieee.org/top-programming-languages/

Source-code Hosting with Ada Build Tools?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Subject: Source-code hosting with Ada build tools?
Date: Fri, 25 Mar 2022 18:56:56 +0200
Newsgroups: comp.lang.ada

I'm planning to move a biggish Ada project from being hosted on my own website to some hosting service, such as GitHub or OSDN. Are there any such services that, in addition to a source-code repository, bug reporting, etc., also offer access to Ada compilers (that is, gnat) for building the SW, ideally on several platforms?

At the moment, my main candidate is OSDN, but they explicitly do not provide any compilers.

TIA for any suggestions, whether with build tools or without.

From: Simon Wright <simon@pushface.org>
Date: Fri, 25 Mar 2022 21:00:58 +0000

Github Actions do this; though I've never set them up for myself.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 26 Mar 2022 21:15:16 +0200

> Github Actions do this; though I've never set them up for myself.

Are you sure that they provide Ada compilers that can be called in an Action?

I tried to find out on the GitHub website, but could not find any list of all the supported languages, and the specific languages they mentioned did not include Ada, and the Search function found nothing about Ada compilation.

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 25 Mar 2022 21:01:31 +0000

> Github Actions do this; though I've never set them up for myself.

AdaCore has one for GNAT.

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 26 Mar 2022 21:04:13 +0000

> Are you sure that they provide Ada compilers that can be called in an Action?

https://github.com/marketplace/actions/ada-actions-toolchain

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 26 Mar 2022 23:46:26 +0200

> https://github.com/marketplace/actions/ada-actions-toolchain

Thanks! Looks like GitHub will be my choice, although I am usually a bit Microsoft-allergic.

From: Luke A. Guest <laguest@archeia.com>
Date: Sun, 27 Mar 2022 16:59:47 +0100

> Thanks! Looks like GitHub will be my choice, although I am usually a bit Microsoft-allergic.

Aren't we all?

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue, 29 Mar 2022 21:38:09 +0300

I have my Ahven library and other things at Sourcehut.org: https://hg.sr.ht/~tkoskine/ahven/

They offer generic build service also. For example see one build log from Ahven: https://builds.sr.ht/~tkoskine/job/675294

The build configurations are Yaml files: https://hg.sr.ht/~tkoskine/ahven/browse/.builds?rev=tip

Of course, the software on the build service is limited to open source operating systems and compilers (Linux, *BSDs, GNAT).

Commercial Ada compilers (like ObjectAda or Janus/Ada) are not supported.

For commercial Ada compilers, I run internal homelab network with Jenkins master on RPi4 and couple of Windows build slaves, which fetch the source code from Sourcehut periodically.

And before starting to use Sourcehut, read the caveats page: https://sourcehut.org/alpha-details/

I also think that Sourcehut doesn't support hosting of "random" binaries, like hand-crafted release tar balls. These kinds of things I locate on a separate virtual server.

Ada-related Tools

HAC v.0.0996

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: HAC v.0.0996
Date: Sat, 22 Jan 2022 01:41:09 -0800
Newsgroups: comp.lang.ada

HAC (HAC Ada Compiler) is a small, quick, open-source Ada compiler, covering a subset of the Ada language. HAC is itself fully programmed in Ada.

Web site: http://hacadacompiler.sf.net/

Source repositories:

#1 svn: https://sf.net/p/hacadacompiler/code/HEAD/tree/trunk/

#2 git: https://github.com/zertovitch/hac

* Main improvements since v.0.095:

- range checks on discrete subtype assignment (:=) and conversion
- short-circuit logical operators: "and then", "or else"
- for S = Scalar subtype: S'First, S'Last, S'Succ, S'Pred, S'Pos, S'Val, S'Image, S'Value, S'Range attributes
- for A = array object or array subtype: A'First [(N)], A'Last [(N)], A'Range [(N)], A'Length [(N)] attributes
- "&", "<", ">", "=", "/=" operators defined for the String type (additionally to HAL.VString type)
- CASE choices admit ranges
- forward declarations for subprograms

Enjoy!

PS: for Windows there is an integrated editor that embeds HAC: LEA: http://l-e-a.sf.net

HAC v.0.1

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: HAC v.0.1
Date: Sat, 14 May 2022 05:35:55 -0700
Newsgroups: comp.lang.ada

[Omitted common info to previous HAC announcements. —arm]

* Main improvements since v.0.0996:

- packages and subpackages are now supported
- modularity: packages and subprograms can be standalone library units, stored in individual files with GNAT's naming convention, and accessed from other units via the WITH clause
- validity checks were added for a better detection of uninitialized variables.

Package examples and modularity tests have been added. Particularly, a new PDF producer package with a few demos is located in the ./exm/pdf directory.

Enjoy!
Gautier

PS: for Windows, there is an integrated editor that embeds HAC:

LEA: http://l-e-a.sf.net

PPS: HAC will be shown at the Ada-Europe conference (presentation + tutorial)

http://www.ada-europe.org/conference2022/

From: Doctor Who <doc@tardis.org>
Date: Sat, 14 May 2022 18:05:55 +0200

Which subset of the Ada language is covered?

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sat, 14 May 2022 22:24:04 -0700

> which subset of the Ada language is covered?

Quoting from ./doc/hac.txt (section "Language subset"):

"The available Ada language subset supported by HAC is so far, roughly, the "Pascal subset", plus tasking, plus packages, less pointers. From a different perspective, HAC supports Ada 83, less pointers, less generics, less unconstrained types, plus a few items from Ada 95 and 2005. Recursion and nested subprograms are supported."

and: "Tasks are implemented, but not working yet."

From: Leo Brewin <leo.brewin@monash.edu>
Date: Sun, 15 May 2022 10:14:41 +1000

I just tested this on macOS Monterey 12.3.1 and it works perfectly out of the box (as expected for Ada code :).

Great work Gautier!

From: Bill Findlay <findlaybill@blueyonder.co.uk>
Date: Sun, 15 May 2022 02:39:36 +0100

> I just tested this on macOS Monterey 12.3.1 and it works perfectly out of the box (as expected for Ada code :)

You beat me to it by an hour!

> Great work Gautier!

Ditto.

HAC v.0.2

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: HAC v.0.2
Date: Sat, 25 Jun 2022 00:43:14 -0700
Newsgroups: comp.lang.ada

[Omitted common info to previous HAC announcements. —arm]

* Main improvements since v.0.1:

- a program run by HAC can exchange data with the program running HAC, through dynamically registered call-backs
  - see package HAC_Sys.Interfacing and demos: src/apps/exchange_native_side.adb src/apps/exchange_hac_side.adb
- the compiler checks that all choices in a CASE statement are covered
- the compiler performs more compile-time range checks and optimizes away useless run-time checks when it's safe to do so.

AdaStudio-2022 Rel. 12/04/2022 Free Edition

From: Leonid Dulman <leonid.dulman@gmail.com>
Subject: Announce: AdaStudio-2022 release 12/04/2022 free edition
Date: Wed, 13 Apr 2022 01:19:51 -0700
Newsgroups: comp.lang.ada

I'm pleased to announce AdaStudio-2022.

It’s based on Qt-6.3.0-everywher opensource (expanded with modules from Qt-5.15: qtgraphicaleffects qtgamepad qtspeech qtx11extras qtwinextras), VTK-9.1.0, FFMPEG-5.1, OpenCV-4.5.5, SDL2-2.0.20, MDK-SDK (wang-bin) Qt6ada version 6.3.0 open source and qt6base.dll, qt6ext.dll (win64), libqt6base.so, libqt6txt.so (x86-64) built with Microsoft Visual Studio 201 x64 Windows, gcc amd64 in Linux. Package tested with GNAT gpl 2020 Ada compiler in Windows 64bit, Linux amd64 Debian 11.1 AdaStudio-2022 includes next modules: qt6ada, vtkada, qt6mdkada, qt6cvada (with face recognition) and voice recognizer.

Qt6Ada is built under GNU LGPLv3 license

https://www.gnu.org/licenses/lgpl-3.0.html.

Qt6Ada modules for Windows, Linux (Unix) are available from Google drive https://drive.google.com/drive/folders/0B2QuZLoe-yiPbmNQRl83M1dTRVE?resourcekey=0-b-M35gZhynB6-LOQww33Tg&usp=sharing

WebPage is https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio/index.html

[List of detailed files omitted. —arm]

The full list of released classes is in "Qt6 classes to Qt6Ada packages relation table.pdf"

The simple manual how to build Qt6Ada application can be read in "How to use Qt6ada.pdf"

If you have any problems or questions, let me know.

Leonid (leonid.dulman@gmail.com)

Generic Image Decoder v.10

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: Generic Image Decoder v.10
Date: Sun, 17 Apr 2022 03:26:25 -0700
Newsgroups: comp.lang.ada

There is a new release of GID - the Generic Image Decoder.

Home page: http://gen-img-dec.sf.net/

Project page #1: http://sf.net/projects/gen-img-dec/

Project page #2: https://github.com/zertovitch/gid

New in V.10

* Added a decoder for the QOI (Quite OK Image) format
* Added an "all RGB" demo

About GID

The Generic Image Decoder (GID) is an Ada package for decoding a broad variety of image formats, from any data stream, to any kind of medium, be it an in-memory bitmap, a GUI object, some other stream, arrays of floating-point initial data for scientific calculations, a browser element, a device...

Animations are supported.

Features

* Standalone (no dependency on other libraries, bindings...)
* Unconditionally portable code: OS-, CPU-, compiler- independent code.
* Multi-platform, but native code built
* Task safe
* Endian-neutral
* Use of generics and inlining at multiple nesting levels for fast execution
* Free, open-source

Currently supported formats are: BMP, GIF, JPEG, PNG, PNM (PBM, PGM, PPM), QOI, TGA.

SparForte 2.5

From: Ken Burtch <koburtch@gmail.com>
Subject: ANN: SparForte 2.5
Date: Wed, 27 Apr 2022 06:00:12 -0700
Newsgroups: comp.lang.ada

SparForte is my Ada-based shell, scripting language and template engine.

Version 2.5 is available from www.sparforte.com.

Changes since 2.4:

New features/examples: 23

Changes: 8

Fixes: 26

Known Issues:

On Raspian Bullseye, the calendar package has rounding errors. Possibly due to increased precision of time values in the kernel.

On FreeBSD 13, "environment corrupt" errors are being reported when the spar command runs another spar command. Possibly due to out-of-data GCC Ada for FreeBSD.

Tab completion does not work correctly on directory names containing spaces.

Change Log can be viewed here: https://www.sparforte.com/news/2022/news_apr2022.html

A summary of new features can be viewed here: https://www.pegasoft.ca/coder/coder_january_2022.html

SparForte is my hobby and is built with the support of volunteers. It is open source and is about 123,000 lines of code. It has been in development since 2001.

GCC 12.1.0

From: Simon Wright <simon@pushface.org>
Subject: ANN: GCC 12.1.0
Date: Wed, 11 May 2022 17:58:26 +0100
Newsgroups: comp.lang.ada

Find GCC 12.1.0 & tools for Intel silicon (will run on M1 silicon under Rosetta) at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-x86_64

Built on High Sierra with Python 3.8 (because Apple have withdrawn 2.7 in Monterey).

GCC 12.1.0 for Apple Silicon (aarch64)

From: Simon Wright <simon@pushface.org>
Subject: [ANN] GCC 12.1.0 for Apple silicon (aarch64)
Date: Fri, 27 May 2022 14:05:19 +0100
Newsgroups: comp.lang.ada

Find at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-aarch64-1

SweetAda 0.10

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: ANN: SweetAda 0.10
Date: Thu, 12 May 2022 14:54:29 -0700
Newsgroups: comp.lang.ada

I've just released SweetAda 0.10.

SweetAda is a lightweight development framework to create Ada systems on a wide range of machines. Please refer to https://www.sweetada.org.

Release notes @ https://www.sweetada.org/release_notes.html.

Downloads available @ https://sourceforge.net/projects/sweetada.

This release comes with a huge cleanup of the whole system, with many changes in all areas. The build system seems pretty efficient and stable, with no redundant actions, and is able to accommodate a large set of configuration.

The profile agrees with Ravenscar, and all platforms tested run OK, albeit many of them in a very simple manner. Interrupt handling is for some CPUs still only a placeholder, but many of them are able to handle at least a simple timer in order to have a raw notion of time.

There is a Monitor module (very exemplary) to do user interaction and the Srecord module that could be used as a built-in tool to execute fragments of code. The Time module should provide basic capabilities in order to manipulate a datetime.

Many other changes, large cosmetic refinements and an improved documentation. Syntax changes to adhere Ada 2012/202x and some generics removed from I/O layers to simplify the code and gain speed.

With SweetAda 0.10, I also provide new toolchains based on GCC 11.3.0 (release-20220429), you can find them at SweetAda home or at SourceForge. QEMU emulators are bumped to 7.0.0 (release-20220504).

Unfortunately, I can no longer provide OS X toolchains due to increasing difficulties in building the software (GCC and LLVM disagree on the syntax of some CPU instructions), and lack of time, sorry. But this shouldn't be a problem since SweetAda should be toolchain-agnostic.

Simple Components v4.62

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components v4.62
Date: Sat, 21 May 2022 12:03:59 +0200
Newsgroups: comp.lang.ada

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

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

Changes to the previous version:

- Pipe stream implementation added;
- GNAT 12.1 bugs worked around in several package, in particular, in GNAT.Sockets.Server;
- Bug fix in Generic_Set procedure Replace, parameter Updated unset;
- Bug fix in Tables.UTF8_Names procedure Replace, parameter Offset unset under circumstances.

PragmAda Reusable Components

From: Pragmada Software Engineering <pragmada@pragmada.x10hosting.com>
Subject: [Reminder] The PragmAda Reusable Components
Date: Wed, 1 Jun 2022 12:23:50 +0200
Newsgroups: comp.lang.ada

The PragmARCs are a library of (mostly) useful Ada reusable components provided as source code under the GMGPL or BSD 3-Clause license at https://github.com/jrcarter/PragmARC.

This reminder will be posted about every six months so that newcomers become aware of the PragmARCs. I presume that those who want notification when the PragmARCs are updated have used Github's notification mechanism to receive them, so I no longer post update announcements. Anyone who wants to receive notifications without using Github's mechanism should contact me directly.

A New Era for Ada/SPARK Open Source Community

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Subject: AdaCore Blog: A New Era For Ada/SPARK Open Source Community
Date: Mon, 6 Jun 2022 01:56:27 -0700
Newsgroups: comp.lang.ada

I am sharing this here for people who might not have seen it yet:

https://blog.adacore.com/a-new-era-for-ada-spark-open-source-community

[The blog post announces the relicensing of several AdaCore libraries as Apache 2.0 and the discontinuation of the Community Edition in favor of the FSF branch distributed through Alire. —arm]

I will be at the Ada-Europe conference next week if someone wants to talk live about these announcements.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 06 Jun 2022 09:03:25 -0700

Thanks for posting this.

I hope this will reduce the differences among the GNAT versions available in the various OS releases; they've been causing headaches for Emacs ada-mode. But probably not.

Alire 1.2.0

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Subject: [ANN] Alire 1.2.0
Date: Mon, 6 Jun 2022 01:57:09 -0700
Newsgroups: comp.lang.ada

The new release is here:
https://github.com/alire-project/alire/releases/tag/v1.2.0

Adalog Components

From: J-P. Rosen <rosen@adalog.fr>
Subject: [Ann] Adalog components
Date: Mon, 27 Jun 2022 18:23:18 +0200
Newsgroups: comp.lang.ada

I have added a new component to ease processing of CSV files. Moreover, I have renamed package Debug to Tracer, there were too many packages called Debug around!

More details from Adalog's components page:

https://adalog.fr/en/components.html

Ada and Operating Systems

Building GNAT-FSF on FreeBSD

From: William <william@sterna.io>
Subject: Building GNAT-FSF on FreeBSD
Date: Sat, 12 Feb 2022 21:09:54 +0100
Newsgroups: comp.lang.ada

I did succeed to build a modern gcc (with Ada-GNAT FSF) on my FreeBSD 13.0 serveur. :-)

Fernando Oleo Blanco was very inspiring to me (NetBSD porting), and I use Simon J. Wright portings to macOS for my Hackingtosh.

So I decided to do it too!

For now I did a quick try with plain gcc «out of the box»: (story short)

1. Install gcc6-aux pkg from FreeBSD port (2014 -- Last Updated on 2022-01-26). (see also http://www.dragonlace.net)
2. get gcc 10.3 src from GNU.org and compile it with gcc6-aux (gnat compiler seems OK)
3. get gcc 11.2 src from GNU.org and compile it with the just installed gcc/gnat 10.3

In the first place I thought it would not be successful ...

Now it's time to build) and run the ACATS 4.1y

I took a look at Simon’s ACATS Testsuite on SourceForge, but I need to understand those automated scripts.

I’d like to parallelise a maximum of ACATS sub-projects in order to reduce time.

WIP!!

See you later, William

From: Simon Wright <simon@pushface.org>
Date: Mon, 14 Feb 2022 09:52:31 +0000

The section "Testing in GCC" in the README tells how to run the tests within the GCC framework that allows parallel running. Note, you'll probably have to hammer C-c to abort a parallel run, the script doesn't respond well to that.

I would have liked to get parallelising working, but those scripts! eww!

From: Simon Wright <simon@pushface.org>
Date: Mon, 14 Feb 2022 11:45:05 +0000!

See this thread:
https://gcc.gnu.org/pipermail/gcc/2018-July/226729.html

[This thread discusses how to interrupt Ada tests, as Ctrl-C fails sometimes. –arm]

I'm not sure, but I think that GCC/Ada folk regard the ACATS (2.6, I think) in GCC as more of a confidence thing (DEC used to call it an IVP, Installation Verification Procedure) than a full check.

[...]

Ada Inside

Controlling ST7789 Screen on a RPi Pico

From: Björn Lundin <b.f.lundin@gmail.com>
Subject: Controlling st7789 screen from Ada on a rpi Pico?
Date: Tue, 15 Feb 2022 22:18:44 +0100
Newsgroups: comp.lang.ada

So, I got my first Raspberry Pico :-)

I also got a 'Pico Explorer Base' device at https://shop.pimoroni.com/products/pico-explorer-base

This thing has a st7789 screen. I got it to work with Python.

Now - I see that there is work done with the Pico and Ada - the https://pico-doc.synack.me seems to be a good place to start.

I wonder if there is any port done already for this screen in Ada? Google points me to some python and some c/c++ implementations (whereof Pimoroni's Github has some)

I also came across uGUI http://embeddedlightning.com/ugui/ which looks interesting. Same question there. Ada-port?

I hesitate to start translating one of the c-libraries - butI probably will when time permits if nothing is already in place.

From: jer...@synack.me <jeremy@synack.me>
Date: Tue, 15 Feb 2022 18:03:26 -0800

The Pimoroni Picosystem uses a ST7789 screen, I have a driver for it in picosystem_bsp: https://github.com/JeremyGrosser/picosystem_bsp/tree/master/src

I didn't implement every feature or video mode that the controller supports, so you may need to modify it to suit your needs.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Wed, 16 Feb 2022 08:19:07 +0100

Perfect - just what I was looking for - thanks.

And thanks for the effort of bringing Ada to the Pico

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Fri, 18 Feb 2022 01:31:20 -0800

> I also came across uGUI <http://embeddedlightning.com/ugui/> which looks interesting

I have an Ada binding [1] for the excellent lvgl GUI library [2]. You can get it from Alire: [3].

It is not in a very beginner friendly shape, but it works. I am trying to do a new version that should be easier to integrate into existing projects.

Don't hesitate to say hello on the Ada Gitter chat if you want a little help setting it up.

[1] https://github.com/Fabien-Chouteau/lvgl-ada

[2] https://github.com/lvgl/lvgl

[3] https://alire.ada.dev/crates/lvgl_ada.html

Ada in James Webb Space Telescope? (Cont.)

[Refer to AUJ 43-1: Ada in James Webb Space Telescope? —arm]

From: 姚飞 <yaofei509@gmail.com>
Subject: Re: is Ada used in James Webb Space Telescope software?
Date: Sat, 23 Apr 2022 02:17:05 -0700
Newsgroups: comp.lang.ada

> Interesting. I hadn't heard of the MA31750 but it appears to be a 16 bit processor that implements the MIL-STD-1750A instruction set(!), which I didn't know about either. Apparently it was made in the 1980s but has since been superseded by SPARC architecture cpu's.

MAS31750 + XGC M1750-Ada is a very wonderful combination, we use them for several large satellites, and they are working in orbit now.

Ada and Other Languages

Comparing Languages wrt Energy, Speed, and Memory Use

From: Jerry <list_email@icloud.com>
Subject: Comparing languages wrt energy, speed, and memory use
Date: Sun, 20 Feb 2022 14:59:29 -0800
Newsgroups: comp.lang.ada

This paper comparing 27 languages with respect to energy use, speed, and memory use is interesting. Of course Ada fares very well.

https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf

It is linked from this Slashdot page which I'm sure is full of useless chatter.

https://developers.slashdot.org/story/22/02/20/0143226/is-it-more-energy-efficient-to-program-in-rust

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Tue, 22 Feb 2022 21:10:15 +0100

I am going to leave a few comments regarding this paper that I believe everybody should know. Most if not all of these points are known and have been discussed pretty much everywhere; but a lot of people still don't know them or decide to not know.

The programs are taken from the Programming Language Benchmark Game. It is a really cool place that has been providing relevant performance data for a lot of languages and comparisons between them.

Here are a few issues:

1. Quite a few languages are not using heavily optimised code. Ada is one of them. Some of those programs are written as direct translations from other languages from people that did not know the target language.
2. Quite a few of those implementations have not been touched in years. Some of the improvements that may have taken place in the language/compiler/tools may not be taken advantage of. For example, the Ada examples are compiled with -gnatNp. Can anybody say what that flag does? x)
3. C/C++/Rust programs are competing on getting the best results. Other languages are lagging behind. For example, Fortran could do much better. For a couple of years, the Fortran community has been improving the code little by little and they have managed to improve their results.
4. There are a few controversies. Some languages are not allowed to use higher performance libraries while others are allowed their stl or equivalent that do actually use the same tools as those libraries. There are a few other examples.

As the very Game page says, do not take the benchmark seriously. But the communities whose languages are on top, they do not care. Ada has been left behind since very few or nobody is actually taking a look at the code and optimising it...

We may want to improve some of these tests as a community :)

Here are some relevant links:

- Benchmark game: https://benchmarksgame-team.pages.debian.net/benchmarksgame/

- Source code: https://salsa.debian.org/benchmarksgame-team/benchmarksgame

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 22 Feb 2022 21:49:25 +0100

> [good remarks snipped]

Let me add another one: this benchmark does not consider the energy (electrical and human) needed to write and debug the program... That could also make a difference for Ada!

Real ecological balance, taking everything into account, is tricky...

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Thu, 24 Feb 2022 08:42:40 +0100

> Here are a few issues:

One issue is Isaac Gouy's clever approach. (Not complaining. I sometimes didn't see the point, though, of adopting another new thing. For example, when a new regex library was introduced (at some point) that wins hands down by using optimization techniques you'd associate with JIT compilers or with data based optimization. Worth knowing about, but how does it help comparing languages when all you can do is link it to your program?)

> 1. Quite a few languages are not using heavily optimised code

Can you be specific? For example, at least one program currently leads by making extensive use of x86 intrinsic ops.

Some use OMP with intrinsic 128bit ops. Does GNAT have a similar parallel loop in the language yet?

> 2. Quite a few of those implementations have not been touched in years.

Yet, some Ada program versions #N+m used to run faster than #N. They now have their speed difference wiped out or even reversed... I see -march=ivybridge now, so the hardware has likely changed.

> For example, the Ada examples are compiled with -gnatNp. Can anybody say what that flag does? x)

GNAT User's Guide explains. (su-p-press and front end i-N-lining)

> 3. C/C++/Rust program are competing on getting the best results. Other languages are lagging behind. For example, Fortran could do much better.

How would Fortran do much better? Can Ada learn from that?

[...]

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Thu, 24 Feb 2022 10:13:46 +0100

> Can you be specific? For example, at least one program currently leads by making extensive use of x86 intrinsic ops.

> Some use OMP with intrinsic 128bit ops. Does GNAT have a similar parallel loop in the language yet?

Yes, take a look at https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/nbody-gnat-2.html

it is taken from the Pascal implementation and uses intrinsics. My point is that some of these programs are not very Ada-like. As far as I remember, there was one ported from Lua.

Ada 2022 will have a parallel keyword. However, it is still not supported in FSF GNAT, which is the one being used. Also, the benchmarks are Ada 2012.

> GNAT User's Guide explains. (su-p-press and front end i-N-lining)

Correct, but that switch has been deprecated for years, it is no longer documented anywhere in the new GNAT releases:

https://gcc.gnu.org/onlinedocs/gcc-11.2.0/gnat_ugn.pdf

> How would Fortran do much better? Can Ada learn from that?

Fortran is using Intel's compiler, which is known to be one of the best. Fortran compilers can much more easily generate SIMD code and parallelise loops automatically if the code is idiomatic.

Also, Fortran was not fourth in the race a while ago. For example Ada overtook Fortran for a small while. December 2018:

https://web.archive.org/web/20181204085050/https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fast.html

Ada is fourth; while it was fifth in April of that same year

https://web.archive.org/web/20180406194535/https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fastest.html

A year later, December 2019, Fortran could be fourth if it were not for that outlier

https://web.archive.org/web/20191225172425/https://benchmarksgame-team.pages.debian.net/benchmarksgame/which-programs-are-fastest.html

These are the current results:

https://benchmarksgame-team.pages.debian.net/benchmarksgame/box-plot-summary-charts.html

Take a look at the evolution of the language podium. It has always been C/C++/Rust, but starting from the fourth position there has been quite a bit of rivalry.

[...]

Some Ada programs could use better algorithms, data structures, more up-to-date syntax and parallelism. Some programs could also be made a bit prettier.

The crux of the issue is that you can pretty much always get peak performance for non-GC languages if you use the same techniques, libraries, algos, state of the art compilers, etc. And in a lot of real world cases, even GC languages are not an issue, see Go, Erlang, Julia, Lisp (SBCL), Nim...

But as someone (I believe it was the dean of TUM (Technische Universität München)) once said: "Everybody knows that rankings are flawed, but it is always better to be on top." The benchmark game is, after all, a game. But some people took it too seriously. It is just like Football hooligans.

From: 25.Bx943 <25bz493@nada.net>
Date: Sat, 26 Feb 2022 22:31:19 -0500

After 30+ years, I started messing around with FORTRAN again. One of the things I noticed in the various help notes online was that programmers were actually comparing the numbers of cycles and executables size for various ways of solving any particular problem.

This sort of thinking is rarely seen these days except in the microcontroller universe - and less even there because the RAM/ROM and speed of those devices has increased.

Ada is another language where overall "efficiency" gets at least some consideration.

With energy costs rising, maybe it's time to see MORE of these discussions and comparisons. Global warming be damned - this is a MONEY issue :-)

Oh, and rising power costs may disappear the crypto sector. Those boxes full of GPUs calculating like mad - the power usage is stupendous. Once the energy-in begins to exceed the value of the Bitcoins-out - it's all over.

From: Robin Vowels <robin.vowels@gmail.com>
Date: Sun, 27 Feb 2022 00:05:48 -0800

> This paper comparing 27 languages with respect to energy use, speed, and memory use is interesting.

Has this anything to do with reality?

What of the design, testing, and maintainability of programs?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sun, 27 Feb 2022 09:56:12 +0100

> What of the design, testing, and maintainability of programs?

There are a couple of obvious problems with this study. First, the same data structures, algorithms, and checks for validity of input and so on, in any imperative language, should give very similar machine code. Robert Dewar famously had a collection of equivalent Ada and C programs that produced identical machine code when compiled with gcc. The kind of differences reported between C and Ada or C++ shows that they are comparing apples to orangutans.

Second, there are hard data that show that, compared to low-level languages like C, Ada requires 1/2 the effort to reach deployment, and 1/40 the effort to correct post-deployment errors. The energy consumption for that additional effort should swamp the kind of small differences during execution that this study concentrates on.

Ruby and Ada

From: Mockturtle <framefritti@gmail.com>
Subject: Ruby and Ada
Date: Sat, 14 May 2022 01:46:06 -0700
Newsgroups: comp.lang.ada

As you can guess, my language of choice is Ada, but for small things (often "fast and dirty") or to extract stuff from text files, I use Ruby which I prefer over its direct competitor (much more popular) Python.

Then I read this [1]

> Its [of Ruby] creator, Yukihiro “Matz” Matsumoto, combined parts of his favorite languages (Perl, Smalltalk, Eiffel, Ada, and Lisp)

This could explain the affinity... (Matz is an Adaist! :-))

[1] https://dev.to/rodmatola/ruby-the-best-language-for-general-automation-gh3

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sat, 14 May 2022 05:53:21 -0700

> [...] for small things [...]

BTW: for the small things you describe, you could be tempted by HAC (see the post about HAC a few hours later) :-)...

From: Sromatic <sriviere17@gmail.com>
Date: Thu, 19 May 2022 00:40:03 -0700

I second that.

We wrote in HAC more than 10K lines of code for about 50 scripts (the biggest ones being 3000 lines, the smallest ones less than 20 lines).

I know Ruby (also very nice for small jobs) but HAC is much better (1) and certainly faster too (2)... We also coded a lot in Bash (there are sysadmin here too :)

(1) One of the great things about HAC is that all HAC code can be compiled by GNAT.

(2) HAC is 7 times faster than Bash

And, recently, HAC handles packages... This allows us to have modularity in the Ada way... HAC is a golden nugget ;)

From: Robin Vowels <robin.vowels@gmail.com>
Date: Sat, 14 May 2022 06:40:44 -0700

> [...] for small things [...]

Whether it's small and dirty or something big, PL/I is a great all-rounder.

When Ada Was the Most Popular Language

From: Nasser M. Abbasi <nma@12000.org>
Subject: The good old days, when Ada was the most popular language
Date: Sat, 28 May 2022 06:46:15 -0500
Newsgroups: comp.lang.ada

Check out this cool video

"Most Popular Programming Languages 1965 - 2019"

https://www.youtube.com/watch?v=Og847HVwRSI

At 1:47

1986. Ada was the most popular programming language! (before C took over)

Who Needs Types? Types Make Code Ugly.

From: Nasser M. Abbasi <nma@12000.org>
Subject: who needs types? Types makes code ugly.
Date: Wed, 1 Jun 2022 22:21:08 -0500
Newsgroups: comp.lang.ada

So Ada had it wrong all the time it seems. From

https://python.land/python-tutorial

In a strongly typed language, you need to specify the exact type of each variable, like String, int, and float. It gets even uglier when objects are involved.

Now let’s look at Python variables. In Python, we can do exactly the same without types:

my_name = "Erik"

my_age = 37

my_salary = 1250.70

As you can see, the Python variant is a lot cleaner and easier on the eyes!

And about possible error, they defend this by saying:

In addition, you’ll find out soon enough during testing and fix the error before the software ever goes to production.

So, I think all that Ada needs is to simply remove all those ugly types from the language and it will become popular like Python is now :)

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 2 Jun 2022 12:47:25 +0200

> In addition, you’ll find out soon enough during testing and fix the error before the software ever goes to production.

Proven beyond a shadow of a doubt by the total absence of security vulnerabilities in production S/W.

From: Ldries46 <bertus.dries@planet.nl>
Date: Thu, 2 Jun 2022 13:47:14 +0200

As someone who has been programming since 1966 I used several different languages, Algol 60, Fortran, Basic, C/C++ and Ada, I like using strong types because the ugliest faults you can create are the ones where you by accident use different types in the input or the output of a formula. Such a fault can work through the complete program and result in very tough error searching. Even when the basic failure is using an integer instead of a real.

From: Ben <ben.usenet@bsb.me.uk>
Date: Thu, 02 Jun 2022 16:02:58 +0100

The terms being using in this thread might need to tightened up a bit because I think you are talking about strong /static/ typing.

Python is strongly typed (though exactly how "strong" is debatable) but the checking is at run-time, so you have to rely on testing rather than the compiler. (I don't know enough about Python's new static type syntax to know how strong that is, but it's optional anyway.)

Also, the OP is talking about removing all those messy types, and that's not necessarily the same as removing type checking, either static type checking or at run-time. Haskell, for example, has strong static type checking, but a lot of Haskell is written without ever using a type because of the language's type inference mechanism.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Thu, 02 Jun 2022 10:28:44 -0700

That's just one tutorial. It likely doesn't reflect the views of most Python programmers. The "python.land" site has no official connection

And for what it's worth, Python 3.5 added support for "type hints".

https://docs.python.org/3/library/typing.html

From: John Perry <devotus@yahoo.com>
Date: Thu, 2 Jun 2022 15:10:57 -0700

> (I don't know enough about Python's new static type syntax to know how strong that is, but it's optional anyway.)

I think you mean "type hints"? The compiler doesn't check even when you specify the types. The typing is available for those who want to use a 3rd party tool to do "stuff" with it. See the note at the top of this page: https://docs.python.org/3/library/typing.html

The Python runtime does not enforce function and variable type annotations.

They can be used by third party tools such as type checkers, IDEs, linters, etc.

> but a lot of Haskell is written without ever using a type because of the language's type inference mechanism.

Correct me if I'm wrong, but do you mean "without ever *specifying* a type"? Several recent languages have taken this up, including Kotlin and Rust, though you have to specify some types.

Even Ada 2022 offers it with the "renames" keyword.

From: Ben <ben.usenet@bsb.me.uk>
Date: Fri, 03 Jun 2022 01:02:27 +0100

> Correct me if I'm wrong, but do you mean "without ever *specifying* a type"?

"Use" was not at all the right word since writing 1+2 obviously "uses" types, but I don't mean specify either since types can be specified simply by writing literals like "abc". I should have said something more technical like "without writing any type signatures".

> Several recent languages have taken this up, including Kotlin and Rust, though you have to specify some types.

Yes, and even C++.

> Even Ada 2022 offers it with the "renames" keyword.

I don't know much about Ada newer than about 1990. I'll take a look...

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Thu, 02 Jun 2022 23:37:51 -0400

>I don't know much about Ada newer than about 1990. I'll take a look...

My condolences -- taken literally, that means you are working with Ada-83 (ANSI/Mil-Std 1815A -- later ISO-8652:1987). The first significant update was Ada-95 (and Air Force funded original GNAT).

From: Ben <ben.usenet@bsb.me.uk>
Date: Fri, 03 Jun 2022 19:13:50 +0100

> My condolences

Thanks, but I'm fine. Knowing was not intended to imply forced to use.

> -- taken literally, that means you are working with Ada-83 (ANSI/Mil-Std 1815A -- later ISO-8652:1987).

That was the only Ada I knew, though I knew about the updates of course. Couldn't find any reference to type inference though. Is there a good place to go for a "summary of changes" between standards?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 3 Jun 2022 21:34:36 +0300

> Couldn't find any reference to type inference though.

As far as I know, the only type inferencing that occur in Ada is in for-loops where the type of the loop parameter variable is inferred from the range or container over which the loop iterates.

> Is there a good place to go for a "summary of changes" between standards?

Each version of the Reference Manual has an "Introduction" chapter that contains a subheading "Language Changes", but those are quite terse. If you can find a "Rationale" document for the version in question that usually has much more information about the changes.

For Ada 95:
https://www.adaic.org/resources/add_content/standards/95rat/rat95html/rat95-contents.html

For Ada 2005:
https://www.adaic.org/ada-resources/standards/ada05/

For Ada 2012:
http://www.ada-auth.org/standards/rationale12.html

For Ada 2022, see the Intro in the RM:
http://www.ada-auth.org/standards/ada2x.html

For Ada 2022 I don't think there is any "Rationale" document (yet), but there are various summaries and introductions, for example:
https://learn.adacore.com/courses/whats-new-in-ada-2022/chapters/introduction.html

From: John Perry <devotus@yahoo.com>
Date: Fri, 3 Jun 2022 13:27:38 -0700

> As far as I know, the only type inferencing that occur in Ada is in for-loops where the type of the loop parameter variable is inferred from the range or container over which the loop iterates.

FWIW I was referring to the optional specification of type in a renames clause, which I first read about here:
https://blog.adacore.com/ada-202x-support-in-gnat

(section "Renames with type inference").

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 3 Jun 2022 19:28:23 -0500

> For Ada 2022 I don't think there is any "Rationale" document (yet)

It is not likely that there will be an Ada 2022 Rationale, as no one has stepped up to write it or pay John Barnes to write it. The closest thing we have is the Jeff Cousins overview, which I can't find an on-line reference to (or my copy, for that matter). I'll check with Jeff and hopefully get more information.

From: Wesley Pan <wesley.y.pan@gmail.com>
Date: Fri, 17 Jun 2022 10:33:07 -0700

> no one has stepped up to write it or pay John Barnes to write it.

How much would it likely cost to pay someone to generate the Ada2022 rationale? Maybe the community can join together to help fund the work?

From: Paul Rubin <no.email@nospam.invalid>
Date: Fri, 17 Jun 2022 13:46:18 -0700

> How much would it likely cost to pay someone to generate the Ada2022 rationale?

Compared to Ada 2012, the 2022 changes look fairly modest.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 17 Jun 2022 21:06:14 -0500

> How much would it likely cost to pay someone to generate the Ada2022 rationale?

Dunno, you'd have to ask John.

I did get a copy of Jeff Cousin's overview that I'll put up on Ada-Auth.org when I get time (probably not until next month).

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Date: Sat, 18 Jun 2022 10:29:02 -0000

>>> It is not likely that there will be an Ada 2022 Rationale

See my follow-up to Randy's June 3 posting quoted above, that I posted on 4 Jun in this newsgroup with subject

"What's new in Ada 2022?" (copied below).

Executive summary:

- John Barnes wrote a 46 page overview on what's new in Ada 2022; it is available as a new appendix in his latest book "Programming in Ada 2012 with a Preview of Ada 2022";
- Jeff Cousin's overview was published in the Ada User Journal (AUJ), and is already available in the online AUJ archive.

Recent addition:

Earlier this week, Tucker Taft presented a very interesting half-day tutorial "Moving up to Ada 2022" at the 26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022), held in Ghent, Belgium. The event was announced in this newsgroup and via various mailing lists and social platforms. Tutorial participants got a nice overview of what's new in Ada 2022 and practical examples of how to use the new features.

(http://www.adaeurope.org/conference2022/tutorials.html#T1).

To conclude, I repeat below my earlier posting with more information on, and pointers to, John's and Jeff's contributions:

Two additional sources of information on Ada 2022 exist:

- the Ada User Journal;

- the new book by John Barnes.

The Ada User Journal (AUJ, http://www.ada-europe.org/auj/home) has published several articles the last few years about the changes in Ada 2022 (then called Ada 220x).

The latest contribution was the above mentioned overview by Jeff Cousins. It is available from the AUJ online archive:

Ada User Journal, Volume 41, Number 3, September 2020

Jeff Cousins: "An Overview of Ada 202x", pp.159-175

http://www.ada-europe.org/archive/auj/auj-41-3-withcovers.pdf?page=43

And then there's of course the new edition of John Barnes' book: "Programming in Ada 2012 with a Preview of Ada 2022"

https://www.cambridge.org/core/books/programming-in-ada-2012-with-a-preview-of-ada2022/AD30275F35CCECB97EAB80ABC32B019C

Previews of the various sections are available on the cambridge.org site mentioned above, such as the first page of the Preface at https://www.cambridge.org/core/books/abs/programming-in-ada-2012-with-a-preview-of-ada2022/preface/21277D825A1D24906949F642B4AD8BE8

That page includes:

"[...] the main chapters describe the 2016 updated version of Ada 2012 in detail. The book concludes with a major appendix describing the key new features of Ada 2022".

(2016 refers to the year of publication by ISO of the Corrigendum which revised Ada 2012.)

I asked John Barnes about the differences between the original "Programming in Ada 2012" and this new book, apart from the extra appendix on Ada 2022. He provided the following info.

“The main changes are twofold.

In the main body, I have updated it to cover all changes introduced by the 2016 corrigendum. I have corrected all known errors (there were quite a lot) and many cross references were wrong.

An idea of the amount of change can be gathered by noting that the original version had just 6 AIs mentioned in the Index. The new edition mentions 55 AIs in the index.

I also updated the text of the main body to use aspects rather than pragmas where relevant.

So the body is now Ada 2016 although we don't usually talk about that.

The new appendix (46 pages) covers all major features of Ada 2022. The associated website also has things such as the full syntax for Ada 2022 in a style matching the book (that's another 30 pages). Also an updated table of the facilities in containers (14 pages). And some worked examples using new features especially using the big integer packages (currently another 14 pages).

Each chapter of the main book ends with a checklist outlining the new features and referring to the appropriate place in appendix 4 where they are discussed.

-- John Barnes, 14 May 2022, with permission"

I hope this helps.

Dirk Craeynest

From: Paul Rubin <no.email@nospam.invalid>
Date: Sat, 18 Jun 2022 15:16:46 -0700

> To conclude, I repeat below my earlier posting with more information on, and pointers to, John's and Jeff's contributions: ...

> I hope this helps.

Yes, thanks, those references are useful for understanding the changes introduced in Ada 2022. I had thought the idea of a formal rationale was different: not just to explain the changes, but also to explain from an authoritative standpoint why the decisions were made. I don't know how important rationales traditionally have been in the Ada world. But, Ada 2012 introduced a much larger set of changes than Ada 2022 did. So I can understand if a rationale was more important in 2012 than in 2022.

I guess if the higher-end Ada community thought that a 2022 rationale was necessary, they would have required it and funded it. As a not-so-serious user or wannabe user, I don't think I need it, but that's just me.

I do notice long after reading "Ada Distilled" that most of the discussions on this group about technical aspects of Ada still baffle me. So I think a more advanced online tutorial would do some good. I believe the current Ada Wikibook is nice for beginners but doesn't cover more advanced topics all that well.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Jun 2022 16:40:06 -0500

> - Jeff Cousin's overview was published in the Ada User Journal (AUJ), and is already available in the online AUJ archive.

Correct. The version of Jeff's overview I have is quite a bit newer than the version from the AUJ, and has many errors corrected. So I would suggest reading that version rather than the original AUJ version (but of course, only once I can get it posted).

Ada Practice

GtkAda for GTK4?

From: Andreas Almroth <andreas@almroth.com>
Subject: GtkAda for GTk4?
Date: Sun, 13 Feb 2022 07:32:23 -0800
Newsgroups: comp.lang.ada

Looking at the excellent support for GTK in GtkAda over the past many years, which I have enjoyed using, I was looking for (aka googling) references to any initial thoughts/work on having GtkAda to also support GTK4.

I know, GTK4 has only been "out" for a little over a year, but it would be interesting to know if anyone is considering doing this. I would be glad to participate, although with limited know-how of the inner workings of GtkAda, but at least testing perhaps.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 13 Feb 2022 17:46:04 +0100

Well, knowing GTK's disastrous history it cannot be "also", it must be either 3 or 4.

GTK 4 breaks basically everything one could ever think of.

To me new features in GTK 4 do not look worth changing the API again, not even useful, just fancy stuff. It seems that GTK team keep on breaking the API rather out of fun than necessity. Instead of hardening the code. GTK 3 is still buggy as hell.

Of course, at some point one will have to migrate, but how about sitting GTK 4 over and going straight to GTK 5? Unless they lose remaining users...

GtkAda is maintained by AdaCore, so it is them [whom you have] to ask.

From: Andreas Almroth <andreas@almroth.com>
Date: Sun, 13 Feb 2022 12:26:11 -0800

[…]

> Of course, at some point one will have to migrate, but how about sitting GTK 4 over and going straight to GTK 5? Unless they lose remaining users...

Well, they might, but it is still based on C, which is easier to interface to from Ada, than say C++ (which I have found cumbersome). Most other GUI frameworks are based on C++, for instance QT. QTAda is as far as I know not maintained (I haven't seen much in a very long time).

[...]

From: Andreas Almroth <andreas@almroth.com>
Date: Mon, 14 Feb 2022 00:45:07 -0800

First, I have to correct myself... Seems Leonid Dulman provides QT5 and QT6 support as part of Ada Studio (google qt6ada). Just saw another post on the adagorge.org re-design, and found QT that way…

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 13 Feb 2022 21:45:57 +0100

> Well, they might, but it is still based on C, which is easier to interface to from Ada, than say C++ (which I have found cumbersome).

Absolutely. C API is a huge advantage. However GTK and stuff is monstrous, practically impossible to handle manually.

GtkAda bindings are generated by a tool designed by AdaCore. This tool might require massive changes when migrating to GTK 4.

I cannot speak for AdaCore, but I think any help will be welcome.

Good luck.

From: Luke A. Guest <laguest@archeia.com>
Date: Sun, 13 Feb 2022 22:35:44 +0000

>> Well, knowing GTK's disastrous history it cannot be "also", it must be either 3 or 4. GTK 4 breaks basically everything one could ever think of.

It's almost like wxAda would've been better...

> Well, they might, but it is still based on C, which is easier to interface to from Ada, than say C++

Can confirm, binding C++ is too easy to burn out on, having done so on wxAda.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sun, 13 Feb 2022 23:46:01 -0800

> GtkAda bindings are generated by a tool designed by AdaCore. This tool might require massive changes when migrating to GTK 4.

I wrote that python script years ago, when the XML files that describe the gtk+ API were actually pretty bad type-wise. The script is full of special cases, and very ugly. I don't think anyone should use it as a basis for binding to gtk 4, it would likely be much better to restart from scratch. I believe the XML files have improved significantly since then, and are used by more language bindings, too, so that could likely be simplified.

From: Andreas Almroth <andreas@almroth.com>
Date: Mon, 14 Feb 2022 00:47:53 -0800

> I wrote that python script years ago [...]

Thanks Emmanuel for your input. Seems it indeed would be a larger effort, and as Dimitry states, perhaps one should wait for the next major release. It will take some time in any event to create the interface binding.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Mon, 14 Feb 2022 20:50:20 +0100

> I wrote that python script years ago

This may be worth mentioning...

Fortran also has a GTK binding [1]. It is also autogenerated with a Python script. As far as I can remember, the change from GTK3 to 4 was not too big. It did obviously require changes and a bit of elbow grease, but they had GTK4 support as soon as it became official. The actual code is present here [2].

This may serve as a comparison or reference for what may be needed. It is obvious that the Fortran people did have a different starting point and Fortran is a different language. I am just including it for reference.

[1] https://github.com/vmagnin/gtk-fortran

[2] https://github.com/vmagnin/gtk-fortran/tree/gtk4/src

What Is the Name of the "|" Symbol?

From: Matt Jaffe <matt.jaffe@gmail.com>
Subject: What is the name of the | symbol?
Date: Fri, 25 Mar 2022 12:04:57 -0700
Newsgroups: comp.lang.ada

In using it in a named association array aggregate, its semantic are "and" --- e.g., some_1D_array := (1 | 3 | 7 => 5, others => 10) sets elements 1 and 3 and 7 to the value 5. In a case statement, its semantics are "or" --- e.g. when 1 | 3 | 7 => ... any of the values 1, 3, or 7 for the case expression will select the ... code for execution. Is there a single name for that symbol (the |) that seems to have different semantics depending on context?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 25 Mar 2022 23:21:37 +0100

ARM 2.1(15/3)

(http://www.ada-auth.org/standards/aarm12_w_tc1/html/AA-2-1.html#I1201) says its name in Ada is "vertical line".

From: Paul Rubin <no.email@nospam.invalid>
Date: Fri, 25 Mar 2022 21:24:32 -0700

The Unicode name is U+007C VERTICAL LINE, alias name vertical bar.

/ is U+002F SOLIDUS, alias names slash and virgule. I haven't heard of the name "solidus" used for symbols other than /.

From: Matt Jaffe <matt.jaffe@gmail.com>
Date: Fri, 25 Mar 2022 12:16:04 -0700

In using it in a named association array aggregate, its semantics are "and" --- e.g., some_1D_array := (1 | 3 | 7 => 5, others => 10) sets elements 1 and 3 and 7 to the value 5. In a case statement, its semantics are "or" --- e.g. when 1 | 3 | 7 => ... any of the values 1or 3, or 7 for the case expression will select the ... code for execution. Is there a single name for that symbol (the |) that seems to have different semantics depending on context?

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Fri, 25 Mar 2022 19:23:19 +0000

How about reading it like this (read with a fixed-width font):

a := (1 | 3 | 7 => 5, others => 10)

if the index is one or three or seven then five else ten fi

Similar syntax appeared in Algol 68. | is frequently used for "alternatives" -- it's just a question of what's being referred to. Here, it's all the alternative indexes that map to a specific value.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 25 Mar 2022 22:03:08 +0200

That is a quirk of natural language, where "and" and "or" are used in non-mathematical ways. You could as well describe the aggregate as saying "if the index is 1 or 3 or 7, the element is 5", and you could describe the case statement as saying "this when-branch is executed when the case selector is 1 and 3 and 7".

As '|' is used in some logical formalisms for disjunction ("or"), and in syntactical notation (BNF) to separate alternatives, I tend to read it as "or".

> Is there a single name for that symbol (the |) that seems to have different semantics depending on context?

For the name, see https://en.wikipedia.org/wiki/Vertical_bar , where indeed "vertical bar" seems favoured. However, I'm pretty sure that I have seen "solidus" used, too, but Wikipedia says that is a synonym for "slash" (/). Wiktionary does not recognize "solidus" as a term for any punctuation mark.

From: Chris Townley <news@cct-net.co.uk>
Date: Fri, 25 Mar 2022 23:24:45 +0000

Probably wrong, but for a Unix user since the last century, I call it 'pipe'

From: Matt Jaffe <matt.jaffe@gmail.com>
Date: Sun, 27 Mar 2022 11:57:48 -0700

> Probably wrong, but for a Unix user since the last century, I call it 'pipe'

Well, non-judgmental type that I am, I'm not going to say you're "wrong", but pipe is the name and usage for that symbol when programming a Unix shell. Its semantics in Ada are quite different, so I don't think calling it pipe quite fits. (So I guess I'm not a pipe-fitter either ;-)

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 26 Mar 2022 00:58:53 +0000

> Probably wrong, but for a Unix user since the last century, I call it 'pipe'

Or bar.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sat, 26 Mar 2022 17:38:59 -0700

> Or bar.

Emacs ada-mode grammar calls it BAR.

From: Matt Jaffe <matt.jaffe@gmail.com>
Date: Sun, 27 Mar 2022 12:01:05 -0700

> Emacs ada-mode grammar calls it BAR.

"Bar" sounds like the best alternative so far; I think that's what I'll use when talking to my students.

Thanks.

From: Chris Townley <news@cct-net.co.uk>
Date: Sat, 26 Mar 2022 02:01:38 +0000

> Or bar.

No that is for after work ;)

Aggregate with (parens) Considered Obsolescent

From: Simon Wright <simon@pushface.org>
Subject: Aggregate with (parens) considered obsolescent
Date: Mon, 11 Apr 2022 17:15:08 +0100
Newsgroups: comp.lang.ada

GCC 12. with the -gnat2022 switch, supports (a large part of) ARM 2022. One of the changes is AI12-0212[1], the use of square brackets [] in array aggregates.

I was surprised to find that the compiler reports the use of parentheses () for array aggregates as obsolescent! To quote PR104751[2],

==========

Compiling

```
procedure New_Syntax is
   T : array (1 .. 5) of Integer;
begin
   T := (1, 2, 3, 4, 5);
end New_Syntax;
```

with -gnat2022 -gnatwj gives

new_syntax.adb:4:09: warning: array aggregate using () is an obsolescent syntax, use [] instead [-gnatwj]

but use of parens is not in Annex J; use of brackets is an option, AARM 202x Draft 32, 4.3.3(49.m).

Having -gnatwj as part of -gnatwa makes this very intrusive.

==========

The fact that it happens with -gnatwa, which is a switch that I suspect quite a lot of us use, will be particularly annoying for those who use -gnatwe (treat warnings as errors) and who want to support multiple compiler releases (for example, the Ada Drivers Library).

The response dismissing the PR suggested using

```
pragma Warnings (Off, "*array aggregate*");
```

and one glimmer of hope is that this can be used as a configuration pragma.

I could remove the problem from macOS releases that I support (sem_aggr.adb:1803..1815), but of course that would lead users into problems when using another GCC 12+ release.

[1] http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai12s/ai12-0212-1.txt?rev=1.29&raw=N

[2] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=104751

Use Clauses and Naming Schemes

[Offshoot from “Unchecked_Deallocation Usefulness”, in AUJ 42-2, 2021. The conversation went into naming preferences in relation to “use” clauses. —arm]

From: Thomas <fantome.forums.tdecontes@free.fr.invalid>
Subject: use clauses
Date: Wed, 13 Apr 2022 01:25:31 +0200
Newsgroups: comp.lang.ada

> For me, a naming scheme that discourages the use of (package) use clauses is a bonus. (Such a scheme makes it easier to avoid use clauses.)

I agree to avoid use clauses.

(I personally prefer Lists.List, like Vincent Marciante - I like Ada.Containers.* naming :-))

> I personally only use "use type" in new code (there's tons of old code for which that doesn't work, of course, but that doesn't change the principle).

what do you think about:

- "use all type" clauses?

- List.Clear? (could you remember me how you call that, please?)

- List.Clear does work only if List is tagged?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 12 Apr 2022 20:05:00 -0500

> what do you think about:

> - "use all type" clauses?

This is OK; I don't use them mainly because I only use features implemented in Janus/Ada, and "use all type" is not yet implemented there.

The fundamental problem with "use" is that it makes everything visible, and then deals with conflicts by making those things invisible again. That's not a problem for overloadable primitive operations, since the profile is included and conflicts only occur when someone has made a lousy design choice (creating a routine with the same name and profile as a primitive) [Most such conflicts come from maintenance when some existing routine is moved to be primitive; in such cases, the original routine simply should be removed.] Since "use all type" only works on overloadable primitives (and things that work rather like primitives), it's fairly safe. One could make an argument that primitive operations should always be visible when the type is (that's not the Ada rule, but arguably it would work better in most circumstances) - and you should always know to look at primitives anyway when trying to find something..

> - List.Clear? (could you remember me how you call that, please?)

For tagged types, you can use prefix notation, so "My_List.Clear" is the easiest. With "use all type List", you can write Clear(My_List). If your objects have well-chosen names, it's not really needed to have the type around for such operations, even when use clauses are in place. Thus, "Clear", not "Clear_List", and that works well even when someone uses everything in sight (of course, they may have a hard time finding where Clear is defined when debugging, but that's their choice).

> - List.Clear does work only if List is tagged?

Right. There are a number of semantic issues for untagged types, the main ones having to do with implicit dereference (which occurs in this notation, as in any other selected_component notation). If you have a prefix of an access type, it gets very messy to determine which dereference is which. And just allowing composite types doesn't work well either: a private type that is completed with an access type would *lose* operations when it had full visibility -- that seems pretty weird.

It originally got limited to tagged types as that was easy to do and didn't have semantic issues. We were going to look at generalizing the prefix notation again (several people asked about it), but no one made a concrete proposal and it never went anywhere for Ada 2022.

Max Line Length Preferences

From: Thomas <fantome.forums.tdecontes@free.fr.invalid>
Subject: max line length
Date: Mon, 18 Apr 2022 23:58:56 +0200
Newsgroups: comp.lang.ada

How do you set your max line length?

Using indentations a lot, I find that 80 is short. but I don't realize how many people I'm going to disturb if I set a greater length, because I don't know all your uses.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Tue, 19 Apr 2022 09:38:00 +0300

I limit lines to 80 characters, because I very often want to use a side-by-side diff of file versions, which means having a window wider than two line-lengths. Text in a 170-character-wide window is still readable, but wider ones are not, for me as an older guy with stiff eye-lenses.

To make do with 80-character lines, I often use local or partial use-clauses, and I divide long calls across many lines, usually having only one parameter per line. By a "partial use clause" I mean, for example, "use Interfaces", when I really need to use Interfaces.C, so I still have to qualify with "C.zzz" but not with "Interfaces.C.zzz".

I also group subsystems into package families (parent and child packages) which means that the children can directly use parent-declared identifiers without qualification.

Other means to keep lines short include using a small indentation step (I now use 3 spaces, but I'm considering changing to 2 spaces) and keeping subprograms short, which also helps the readability.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 21 Apr 2022 16:34:04 -0700

> I limit lines to 80 characters, because I very often want to use a side-by-side diff of file versions,

I prefer top/bottom diff, partly for this reason.

But my monitor can easily display 240 characters across. And I have good glasses.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Tue, 19 Apr 2022 21:30:38 +0200

> how do you set your max line length?

I use the Preferences menu selection in my editor. But that's probably not what you intended to ask. I set mine to 132 characters.

> using indentations a lot, i find that 80 is short.

When I started out, source lines were limited to 80 columns because that was the length of punched cards, but the line printers could print 132 columns. In the 1980s printing switched from 14 x 11 inch paper in line printers to 8.5 x 11 inch paper, but it was still possible to print 132 characters in landscape mode, so that's what I used if I had an editor that could handle long lines easily (screens were not large enough or high enough resolution to be suitable for reading programs, so I still tended to print them when that was needed. Today printing is not needed much, but I continue to use 132 columns. If others want a different line length they may reformat it.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 19 Apr 2022 22:07:20 +0200

> how do you set your max line length?

72. I used to program in FORTRAN on punched cards. (:-))

These days I use 3 split GPS Windows side by side.

Then I am using the "use" clause, so I do not need a thousand of characters to just write Z := X + Y; (:-))

> using indentations a lot, I find that 80 is short.

Refactor the code and use local subprograms.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 21 Apr 2022 16:31:56 -0700

> how do you set your max line length?

120 chars; I assume readers have a big display like mine.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 22 Apr 2022 02:57:08 -0500

> I prefer top/bottom diff, partly for this reason.

The diff program I use can scroll sideways if necessary, and so can every editor I've used since 1985, so this isn't generally an important concern. The Janus/Ada source used a "soft" limit of 80, mainly because that's what terminals and PCs displayed back then, but we never broke lines just for that reason. Typically, the indent is more than the overrun anyway (so that actual text never exceeded 80 characters). Of course, one has to break really long calls, like the call to create a window in Claw (which usually has a dozen or so parameters).

Aspect Location in Expression Function

From: Blady <p.p11@orange.fr>
Subject: Aspect location in expression function.
Date: Sat, 14 May 2022 13:47:28 +0200
Newsgroups: comp.lang.ada

I'm puzzled when I want to change a function body with aspects to an expression function, for instance:

```
function Length (S: Some_Tagged_Tyoe)
return Natural
  with Pre => S.Valid
  is
  begin
  return S.Length;
  end;
```

have to be changed in:

```
function Length (S: Some_Tagged_Tyoe)
return Natural
  is (S.Length)
  with Pre => S.Valid;
```

The location of the aspect has moved to the end.

I'd like simply replace the begin block by the expression, as:

```
function Length (S: Some_Tagged_Tyoe)
return Natural
  with Pre => S.Valid
  is (S.Length);
```

What could be any reasons not to permit it?

From: J-P. Rosen <rosen@adalog.fr>
Date: Sat, 14 May 2022 17:40:03 +0200

What you say is logical if you think of an expression function as a body; however, it is more like a specification (it can appear in a package spec, although it can complete a specification), so the place where the aspect appears makes sense. And it would be confusing to allow the aspect in two different places. It is the same for separate bodies of subprograms.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 23 May 2022 23:05:12 -0500

> What you say is logical if you think of an expression function as a body; however, it is more like a specification

To make a functioning :LR grammar for Ada, I *had* to allow the aspect specification in both places, and then make one of them illegal. Which is more work than just allowing in either place. So I guess it is a matter of perspective. :-)

To the OP: we discussed placement of aspect specifications ad-nauseam, as issues like this always were coming up. There is no consistent rule that really works well, because one does not want small things following large sets of aspect specs -- they can get lost and overlooked.

For instance, one puts aspect specifications after "is abstract" as otherwise that could be lost after a lengthy precondition expression (and it's too important to be lost). See how that could happen in the following (illegal) declaration:

```
procedure P (A, B ,,,)
   with Pre => <very long expression that extends over several lines here>
   is abstract;
```

So something like this (and "is null" as well) require the Pre at the end:

```
procedure P (A, B ,,,)
   is abstract
   with Pre => <very long expression that extends over several lines here>;
```

Expression functions generally follow the same rules as the older null procedures, thus they ended up with the same positioning. It's not as obvious a case here, since the return expression can also be long, but we thought it should be consistent.

BTW, I don't think there ever is a reason to go from [a function –arm] with a normal body to an expression function (assuming the body is legal). A normal body is more readable and less of a hassle during maintenance. The advantage of an expression function is to use it in places where a regular body is not allowed and/or just to be lazy writing the body - neither of which would ever require changing *to* an expression function. Maintenance might require changing *from* an expression function if the body has gotten too complex (for instance, needs a variable declaration), but that generally will require moving the function as well so "ease" of doing so isn't very relevant.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Tue, 24 May 2022 20:24:14 +0200

> For instance, one puts aspect specifications after "is abstract" as otherwise that could be lost after a lengthy precondition expression (and it's too important to be lost).

Isn't this emphasis on "is abstract" losing the very point of abstraction?

> See how that could happen in the following

> (illegal) declaration:

> procedure P (A, B ,,,)

> with Pre => <very long expression that extends over several lines here>

> is abstract;

Who cares to see "is abstract" if P is in a spec? The implementer, I guess, but the client? Less so.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 25 May 2022 00:20:50 -0500

> Who cares to see "is abstract" if P is in a spec? The implementer, I guess, but the client? Less so.

Any client that needs to declare an extension (pretty common in OOP), especially as "abstract" routines mostly are used with root types (and interfaces). I suppose you could "program by error" and just let the compiler complain if you don't give a body for something abstract, but it's generally recommended to know what you're doing and not just try to make the compiler happy.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Wed, 25 May 2022 20:45:58 +0200

> Any client that needs to declare an extension (pretty common in OOP),

Another, dare I say, more frequent way of being a client of a type is being a caller of the type's subprograms, such as P, rather than being an implementer of a type's concrete behavior. (The two can overlap, but I'm thinking of the more frequent human clients here :)

A case I'd single out is a type that comes with a factory F. I'd expect the associated type T to be abstract. This goes without saying! ;-) A client needs to know the "behavioral" interface of T and also that of F. The "is abstract" then remains as helpful language technology, but as seen inside the factory.

(So, I'd put "is abstract" last.)

> especially as "abstract" routines mostly are used with root types (and interfaces). I suppose you could "program by error"

Not design errors, but mechanical errors duly output by the compiler. The programmer will be programming by "following the language's rules". IDEs and compilers will assist the programmer who is implementing an abstract type. For example, the usual IDE has this suggestion following its compiler's error message:

 Fix: "Add unimplemented methods"

(for)

 Error: "The type must implement[!] the inherited abstract method ..."

The IDE will do so if you answer "Yes" and programmers can provide their own adjustments to template text that this mechanism will be using. Thus, again, programmers can involve useful language technology in a template's text. I remember some Ada tools offering similar features.

What Is X'Address of an Empty Array?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: What is X'Address of an empty array?
Date: Tue, 31 May 2022 14:19:24 +0200
Newsgroups: comp.lang.ada

I have a language lawyering question. According to ARM X'Address is the address of the first array element. What is the address of an empty array?

In the case of an array with bounds it could be the address following the bounds.

But what about a definite empty array? Of zero length (and presumably zero size). Would the compiler have to invent some address?

P.S. With GNAT:

```
type NUL is array (1..0) of Integer;
S : NUL;
```

S'Size is 8 and it has some address that holds the byte.

Talking about the dark matter in our Universe. This is what empty arrays are constructed of! (:-))

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 31 May 2022 16:35:49 -0500

Aliased objects should never have zero-size in Ada (or, at least, ever be allocated at the same address). I believe that is

because of the required result of the equality operator. Specifically:

```
type NUL is array (1..0) of Integer;
A, B : aliased NUL;
type PNul is access Nul;
PA : PNul := A'Access;
PB : PNul := B'Access;

if PA = PB then
    Report.Failed ("Access to two distinct
     objects cannot be equal");
end if;
```

If an object is not aliased, it is undefined whether 'Address will work reliably with it (it probably does in GNAT, it might not in Janus/Ada, etc.) If the objects ARE aliased, then 'Address works essentially the same as 'Access.

I personally find this a bit of overspecification in Ada, but since zero-size objects are unusual, no one has thought it worth going through the effort to change. (And of course such a change would complicate static analysis.) We (the ARG) did discuss this topic at one point (I don't have the AI number at hand).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 1 Jun 2022 14:49:25 +0200

> I personally find this a bit of overspecification in Ada

It is actually rather nice.

I recently stumbled upon code:

```
A (A'First)'Address
```

in C bindings, when the other side has something like:

```
const double * a, int len
```

It of course fails in the marginal case when the array is empty. But A'Address seems to never do.

Modern Syntax for Complex Conditions

From: Matt Borchers <mattborchers@gmail.com>
Subject: Ada needs some modernization
Date: Tue, 31 May 2022 10:54:46 -0700
Newsgroups: comp.lang.ada

Throughout my career, I often find myself writing code similar to:

if (A and B) or else (not A and C) then...

and I always wished there was a better and clearer way to write this in Ada. Then along came if expressions. But, if expressions don't help that much with readability although it is arguably simpler:

if (if A then B else C) then...

What amendment can we suggest to the Ada syntax so the if expression be better written when used in an if statement? I know other languages support this and it often looks like A ? B : C or something similar. That's certainly not Ada-like IMO, but I can't think of something better. These same languages often also have a null check operator A ?? B (where A and B are access types of the same Type) such that if A is not null then A is returned otherwise B is returned. So useful and helpful!

Again, I often find myself writing a loop to search for something and then performing one or another action depending on the success of the search. This almost always requires some externally defined variable, like:

```
--assuming arr'First is not Integer'First
found := arr'First - 1;
for i in arr'Range loop
   if arr(i) = match then
      found := i;
      exit;
   end if;
end loop;
if found in arr'Range then
   --do something A
else
   --do something else B
end if;
```

Of course I could move the "do something A" into the if block within the loop, but I still need to know if I must run the alternate code afterward. It would be nice to avoid having to create a variable just to indicate the success state or indexing location found. Maybe something like:

```
for i in arr'Range loop
   if arr(i) = match then
      --do something A
      exit;
   end if;
then
   --do something else B
end loop;
```

The "then" part only executes after the loop terminates normally, i.e. only when the loop does NOT exit early by "exit" or "return" statement.

I think syntax enhancements like these could go a long way to making Ada feel like it is at least keeping up with modern languages and I think current programmers expect "ease-of-use" syntax from today's languages. Other contemporary modernized languages have taken ideas from Ada, but Ada has not continued to pioneer ideas as quickly. Perhaps that's by choice or design.

Thoughts?

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Tue, 31 May 2022 12:05:48 -0700

In your proposal, the "do something else B" appears before "end loop", which is not a very intuitive way to indicate a statement happening *after* the loop.

I suspect there is room for improvement...

Perhaps you would like to show an equivalent piece of code in a what you call a "modern language"?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 31 May 2022 21:55:05 +0200

> if (A and B) or else (not A and C) then...

> if (if A then B else C) then...

Not the same. In the former A may be computed twice.

> What amendment can we suggest to the Ada syntax so the if expression be better written when used in an if statement?

I newer felt it necessary. To me much more aggravating is code that combines test/allocator with renaming, i.e.

```
if P /= null then
   declare
      X : T renames P.all;
   begin
      ...
   end;
end if;

------------

if X in T'Class then
   declare
      XX : T'Class renames T'Class (X);
   begin
      ...
   end;
end if;

-----------

   P : access T'Class := new S;
   X : S renames S (P.all);
```

If one could come up with some syntax for if-then-declare and new-then-declare that would cover a lot of cases.

> I know other languages support this and it often looks like A ? B : C or something similar. That's certainly not Ada-like IMO, but I can't think of something better. These same languages often also have a null check operator A ?? B (where A and B are access types of the same Type) such that if A is not null then A is returned otherwise B is returned. So useful and helpful!

Not in a strongly typed language IMO.

[...]

> Maybe something like:

> for i in arr'Range loop

>   if arr(i) = match then

>   --do something A

>   exit;

>   end if;

> then

>   --do something else B

> end loop;

I usually use a nested function, e.g. search with a fallback:

```
function Get_Me_Something return
Element is
begin
  for I in arr'Range loop
    if Arr (I) = match then
      return Arr (I);
    end if;
  end loop;
  return Default;
end Get_Me_Something
```

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Tue, 31 May 2022 17:46:04 -0500

>What amendment can we suggest to the Ada syntax [...]? I know other languages support this and it often looks like A ? B : C or something similar. That's certainly not Ada-like IMO, but I can't think of something better.

Which is the rub. Ada is *not* about clever operators that hardly anyone knows what they do. Indeed, the original proposal for Ada 2012 had an "implies" operator. But we quickly found out that there are many people that don't know off-hand what function an implies operator does. We were pretty sure that every Ada programmer would understand an if expression.

Note that pretty much the only place that you should almost never use an if expression is in the choice of an if statement. If you already can write an if statement, you don't need an if expression! If expressions exist to make initializations and assertions like (Pre/Post) easier to write.

So I would never write your expression in the first place (either of them). I'd write something like:

```
if A then
   if B then.
   else
  end if;
else
   if C then
   else
  end if;
end if;
```

The contents of the arms should be short anyway, and typically will just be a procedure call (and possibly some debugging, which is way easier if the conditions are kept simple).

>null check operator A ?? B [...] such that if A is not null then A is returned otherwise B is returned. So useful and helpful!

Again, "utility" is not the criteria for Ada, rather understandability for future maintainers is the primary criteria. The last thing we need is a bunch of fancy but little used operators that [leave] someone cold when reading some unfamiliar code. (Yes, of course you can look them up on-line, but stopping to doing so necessarily breaks your train of thought.)

And this construct fits nicely into an if expression, with no magic:

```
(if A /= null then A else B)
```

and this extends nicely to more likely cases:

```
(if A /= null then A elsif B /= null then B
 else raise Program_Error)
```

Personally, I don't believe I've ever written something where such an operator would be useful; one needs to check everything for null (you can't usually can't assume B is nonnull, either). And the fall backs are generally more complex than using some other object. Moreover, probably A should have been declared null-excluding so it doesn't need to be tested in the first place. :-)

> I often find myself writing a loop to search for something and then performing one or another action depending on the success of the search. ...

>for i in arr'Range loop

> if arr(i) = match then

> --do something A

> exit;

> end if;

>then

> --do something else B

>end loop;

>The "then" part only executes after the loop terminates normally, ...

In Ada terms, an exit *is* normal completion, so you would need some different terminology.

> i.e. only when the loop does NOT exit early by "exit" or "return" statement.

We've discussed the "continue" statement multiple times, and have always ended up deciding that we are better off without it. (We've also discussed allowing "exit" from blocks, but that turns into a mess when blocks and loops get mixed, at least if one wants the code to do the same thing in Ada 2012 and in future Ada.)

We've essentially decided that it is better to use a goto in such rare cases. The case you show above is similar.

```
for i in arr'Range loop
   if arr(i) = match then
      --do something A
      goto Loop_Finished;
   end if;
end loop;
-- We get here if the search item is
--  not found:
--do something else B
<<Loop_Finished>> null;
```

Remember that every feature added to a language adds costs in implementation, documentation, and in tools (analysis, checkers, etc.). A feature needs to be quite useful in order to make the cut.

Aside: in the case above, I've usually written such loops like:

```
for i in arr'Range loop
   if arr(i) = match then
      --do something A
      exit;
   elsif i = arr'Last then
     --do something else B
     exit; -- Not really needed, but clearer
           -- what is going on.
   end if;
end loop;
```

I've never been that happy with the duplication of the termination condition, but this avoids any extra objects or any gotos.

If I was going to try to fix your problem with a language feature, I'd probably try to define an attribute to avoid needing to duplicate the termination condition. Something like:

```
Loop_Name: for i in arr'Range loop
   if arr(i) = match then
      --do something A
      exit Loop_Name;
   elsif i = Loop_Name'Range'Last then
     --do something else B
     exit Loop_Name; -- Not really needed,
     --but clearer what is going on.

   end if;

end loop Loop_Name;
```

(We probably would allow 'First and 'Last in such a case.) But this technique doesn't really work with user-defined iterators (which don't necessarily have a defined end), and I'm unsure if it is important enough for another whistle.

>"ease-of-use" syntax from today's languages.

Ada has *never* been about "ease-of-use". It is about readability, maintainability, and understandability. (See the “Design Goals” in the Introduction -- http://www.ada-auth.org/standards/2xrm/html/RM-0-2.html.)

Enhancing readability might also enhance ease of use (for instance, user-defined literals, target name symbols, and user-defined indexing all were added to enhance readability by avoiding duplicative text that provides little information), but it is never a primary goal for an Ada feature.

>Other contemporary modernized languages have taken ideas from Ada, but Ada has not continued to pioneer ideas as quickly.

This is not true. Ada pioneers ideas all the time (see delta aggregates, aggregate iterators, the target symbol, parallel stuff, etc. from Ada 2022). What Ada does not do is waver from its core goal of readability and maintainability. So we don't waste time with tiny features that are more likely to harm readability and understandability than help. (Admittedly, what features are really necessary and which are just nice to have is always a personal choice.) Additionally, Ada has always been designed with a "building-block" approach, so we don't provide (say) a semaphore, but rather the tools (the protected type) to write one (and many other constructs). An if expression is a building block; funny boolean operators with limited uses are not.

I personally am not the least bit interested in worrying about ease-of-use gadgets in other languages. If programmers need such gadgets to be comfortable, they probably don't have the right mindset to be great Ada software engineers in the first place. Saving a few characters in a few expressions simply does not matter when compared to the effort needed to define and document a good data abstraction (for instance, an abstract data type and package).

There *are* features that probably would not interfere with Ada goals of readability. One of them that comes up periodically is an "at end" clause so one could write final wishes for a block/subprogram/package without writing a bunch of exception handlers (which doesn't work in the case of abort!) or one-time use controlled types. I'm sure there are others.

And certainly other languages have interesting features that Ada should steal, the Rust owned access types would be an obvious example. (Don't get me started on why Ada 2022 does not have those.) But "ease-of-use" is not interesting, at least when it does not make readability better. (I want people to replace "and" and "or" with if expressions as much as possible, as those are much more understandable. No more operators please!)

Randy.

P.S. Man, did I spend a lot more time than I planned answering this. I hope it helps.

From: John Mccabe <john@mccabe.org.uk>
Date: Wed, 1 Jun 2022 00:24:24 -0700

> P.S. Man, did I spend a lot more time than I planned answering this. I hope it helps.

FWIW, I thought it was valuable. As I read through it I was constantly thinking of how I wish the people tweaking C++ (which, for various reasons, I'm using now) would take the same attitude, rather than trying to feed their own egos by adding all sorts of random rubbish that, due to the current 3 year cycle, tends also to be either temporary or half-baked random rubbish!

Thank you, Randy!

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Wed, 1 Jun 2022 21:00:31 +0200

> What amendment can we suggest to the Ada syntax [...]

What you call "modernization" looks to me a lot like "repeating mistakes that Ritchie made over 50 years ago".

"A ? B : C"? Or is it "A : B ? C"? If only there were a less cryptic, easier to remember and understand way to express it. Something like "(if A then B else C)", for example.

"A ?? B" might be "useful and helpful" if you use (or think in) a language with pointers to objects everywhere, but in a language where such pointers are never needed, like Ada, it is neither, especially since a conditional expression would handle it just fine if it were ever needed.

> I often find myself writing a loop to search ...

When you write something for a second time, it's a signal to create a subprogram or package to avoid writing it a third time.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Thu, 2 Jun 2022 07:56:53 +0200

> What amendment can we suggest to the Ada syntax so the if expression be better written when used in an if statement?

I would try to fix the problem at where it is caused: ad hoc, unnamed logical predicates! Syntactic sugar won't make these go away.

All those Boolean expressions have meaning, I suppose. The meanings could be given a name. There would be facts, about A, B and C, that make your statement true, some not. What does it state?

Compare this assembly of variables

((A and B) or else ((not A) and C)))

to a lambda expression or to a state machine. Similar? It is the lowest level of computation using a high level language.

> Again, I often find myself writing a loop to search for something and then performing one or another action depending on the success of the search.

Again, there is an algorithm, typically Find_the_First, that will return an index (or cursor). I'd use the return value in a conditional.

From: Brad Moore <bj.mooremr@gmail.com>
Date: Fri, 10 Jun 2022 09:38:35 -0700

> if (A and B) or else (not A and C) then...

> if (if A then B else C) then...

I agree with the other comments, and in a case like this, I might consider writing an expression function to improve readability.

Using cryptic letters for Booleans makes it difficult to assign a name to the expression function, but if you apply it to a less generic example, this becomes easier to do.

For example, if A is renamed to Weekday, B means (time < 9:00pm), and C means (time < 6:00pm) you could write:

```
function Shopping_Mall_is_Open return
Boolean is (if Weekday then
Earlier_than_9_PM else
Earlier_than_6_PM);
```

Then your other code would simply be,

```
if Shopping_Mall_is_Open then ...
```

Brad

Problems Using Generic_Dispatching_Constructor

From: Mark Lorenzen <mark.lorenzen@gmail.com>
Subject: Problems using Generic_Dispatching_Constructor
Date: Wed, 1 Jun 2022 04:36:02 -0700
Newsgroups: comp.lang.ada

The generic function Ada.Tags.Generic_Dispatching_Constructor is defined as:

```
generic
   type T (<>) is abstract tagged limited
private;
   type Parameters (<>) is limited private;
   with function Constructor (Params: not
   null access Parameters) return T is
   abstract;
function
Ada.Tags.Generic_Dispatching_Constructor
(The_Tag : Tag; Params  : not null access
Parameters) return T'Class;
```

This gives us some problems when calling an instance of Ada.Tags.Generic_Dispatching_Constructor when the Params parameter is an in-mode parameter of a function e.g.:

```
function Make (From_Params : in P) return
T'Class
   is
      function Make_T_Class is new
      Ada.Tags.Ada.Tags.
      Generic_Dispatching_Constructor
      (T => T, Parameters => P,
       Constructor => ...);
```

```
begin
  ...
  return Make_T_Class
         (Some_Tag, P'Access);
  end Make;
```

This results in a compile-time error:

error: access-to-variable designates constant

Why is function Ada.Tags. Generic_Dispatching_Constructor defined as:

```
function
Ada.Tags.Generic_Dispatching_Constructor
(The_Tag : Tag; Params  : not null access
Parameters) return T'Class;
```

and not as e.g (note the access-to-constant type):

```
function
Ada.Tags.Generic_Dispatching_Constructor
(The_Tag : Tag; Params  : not null access
constant Parameters) return T'Class;
```

I guess we could declare function Make as (note the in-out mode):

```
function Make (From_Params : in out P)
return T'Class
```

But this is horrible as functions should never ever have in-out or out-mode parameters (or side effects in general).

Why are access types used at all?

Is there another workaround?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 1 Jun 2022 14:42:40 +0200

> Why are access types used at all?

Parameters are kind of a factory object, you want to have the factory mutable.

> Is there another workaround?

In my practice I never had a case when I could obtain the tag needed for generic dispatching constructor. All my designs ended up with a mapping

key -> constructing function

with an explicit registering the type in the mapping.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 1 Jun 2022 16:25:16 -0500

>> Why are access types used at all?

We needed this usable to implement dispatching stream attributes (the generic dispatching constructor was intended to be a user-definable generalization of the mechanism of the class-wide stream attribute). The stream attributes probably used access types because "in out" parameters were not allowed for functions when they were invented. (So mistakes piled on mistakes. :-)

> Parameters are kind of a factory object, you want to have the factory mutable.

Right. For instance, consider a factory where each object gets a unique id while being constructed. You would want to update the Next_Id component at the end of each construction.

>> Is there another workaround?

> In my practice I never had a case when I could obtain the tag needed for generic dispatching constructor. All my designs ended up with a mapping

> key -> constructing function

>

> with an explicit registering the type in the mapping.

Right. Generally, one uses a mapping of some sort of key or menu choice or whatever to tags. If you aren't adverse to a giant case statement, then you might as well call the constructor directly. (And if you are willing to use access-to-functions, you don't need OOP at all.) So this "factory" is mostly a bone for OOP purists.

The one exception is the case where you have an external tag as the key, since you can get the tag from that directly. But even that is really a mapping (one built by the implementation).

Conference Calendar

Dirk Craeynest

KU Leuven, Belgium. Email: Dirk.Craeynest@cs.kuleuven.be

This is a list of European and large, worldwide events that may be of interest to the Ada community. Further information on items marked ♦ is available in the Forthcoming Events section of the Journal. Items in larger font denote events with specific Ada focus. Items marked with ☺ denote events with close relation to Ada.

The information in this section is extracted from the on-line *Conferences and events for the international Ada community* at http://www.cs.kuleuven.be/~dirk/ada-belgium/events/list.html on the Ada-Belgium Web site. These pages contain full announcements, calls for papers, calls for participation, programs, URLs, etc. and are updated regularly.

The COVID-19 pandemic had a catastrophic impact on conferences world-wide. Some events are still planned to be held "virtually", and others in "hybrid" mode, also for wider dissemination. Where available, the status of events is indicated with the following markers: "(v)" = event is held online, "(h)" = event is held in a hybrid form (i.e. partially online).

2022

July 03-07 — 22nd **International Conference on embedded computer Systems: Architectures, MOdeling and Simulation** (SAMOS'2022), Pythagorion, Samos Island, Greece. Topics include: advances in systems efficiency in various domains; novel architectures and computing methodologies and solutions for accelerating applications in various embedded domains; software tools, compilation techniques and optimizations, and code generation for reconfigurable architectures; embedded parallel systems and multiprocessor systems-on-chip; application level resource management of multi-core architectures; all design processes for embedded systems; design languages; performance; reliability; specification languages and models; system-level design, simulation, and verification; MP-SoC programming, compilers, simulation and mapping technologies; profiling, measurement and analysis techniques; (design for) system adaptivity; testing and debugging; etc.

July 05-08 (h) — **Software Technologies: Applications and Foundations** (STAF'2022), Nantes, France.

- July 04-08 — 16th **International Conference on Tests And Proofs** (TAP'2022). Topics include: many aspects of verification technology, including foundational work, tool development, and empirical research; the connection between proofs (and other static techniques) and testing (and other dynamic techniques); verification and analysis techniques combining proofs and tests; program proving with the aid of testing techniques; deductive techniques supporting the automated generation of test vectors and oracles, and supporting novel definitions of coverage criteria; program analysis techniques combining static and dynamic analysis; testing and runtime analysis of formal specifications; verification of verification tools and environments; applications of test and proof techniques in new domains, such as security, configuration management, learning; combined approaches of test and proof in the context of formal certifications (Common Criteria, CENELEC, ...); case studies, tool and framework descriptions, and experience reports about combining tests and proofs; etc.

July 05-08 (h) — 34th **Euromicro Conference on Real-Time Systems** (ECRTS'2022), Modena, Italy.

☺ July 07-08 — 15th **International Symposium on High-Level Parallel Programming and applications** (HLPP'2022), Porto, Portugal. Topics include: high-level parallel programming, its tools and applications, such as high-level programming and tools, automatic code generation for parallel programming, model-driven software engineering with parallel programs, high-level programming models for heterogeneous/hierarchical platforms, applications of parallel systems using high-level languages and tools, formal models of timing and real-time verification for parallel systems, etc.

☺ August 22-26 (h) — 28th **International European Conference on Parallel and Distributed Computing** (Euro-Par'2022), Glasgow, Scotland, UK. Topics include: all flavors of parallel and distributed processing, such as compilers, tools and environments, scheduling and load balancing, theory and algorithms for

	parallel and distributed processing, parallel and distributed programming, interfaces, and languages, multicore and manycore parallelism, etc. Deadline for early registration: July 14, 2022.
☺ August 23-25 (h)	28th IEEE **International Conference on Embedded and Real-Time Computing Systems and Applications** (RTCSA'2022), Taiwan. Deadline for registration: August 19, 2022.
September 04-07 (h)	17th **Federated Conference on Computer Science and Information Systems** (FedCSIS'2022), Sofia, Bulgaria. Deadline for submissions: July 1, 2022 (Data Mining Competition papers).
September 06-09	41st **International Conference on Computer Safety, Reliability and Security** (SafeComp'2022), Munich, Germany. Topics include: development, assessment, operation and maintenance of safety-related and safety-critical computer systems; formal methods for verification, validation and fault tolerance; safety/security co-engineering and risk assessment; testing, verification and validation methods and tools; qualification, assurance and certification methods and tools; cyber-physical threats and vulnerability analysis; safety and security guidelines, standards and certification; etc. Domains of application include: railways, automotive, space, avionics and process industries; highly automated and autonomous systems; telecommunication and networks; safety-related applications of smart systems and IoT; critical infrastructures; medical devices and healthcare; surveillance, defense, emergency and rescue; logistics, industrial automation, off-shore technology; education and training; etc.
September 12-14	15th **International Conference on the Quality of Information and Communications Technology** (QUATIC'2022), Talavera de la Reina, Spain. Topics include: all quality aspects in ICT systems engineering and management; related to the specification, design, development, operation, maintenance and evolution of ITC systems; quality in ICT process, product, and applications domains; practical studies; etc. Deadline for registration: September 5, 2022.
☺ September 14-16	27th **International Conference on Formal Methods for Industrial Critical Systems** (FMICS'2022), Warsaw, Poland. Part of CONFEST'2022. Topics include: case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; methods, techniques and tools to support automated analysis, certification, debugging, descriptions, learning, optimisation and transformation of complex, distributed, real-time, embedded, mobile and autonomous systems; verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability (e.g., scalability and usability issues); impact of the adoption of formal methods on the development process and associated costs; application of formal methods in standardisation and industrial forums.
September 20-22	1st **Summer School on Security Testing and Verification 2022**, Leuven, Belgium. Topics include: static and dynamic security testing; software verification; security by design; etc. Deadline for early registration: July 31, 2022.
September 20-22 (h)	21st **International Conference on Intelligent Software Methodologies, Tools and Techniques** (SOMET'2022), Kitakyushu, Japan. Topics include: state-of-art and new trends in software methodologies, tools, and techniques; software methodologies, and tools for robust, reliable, non-fragile software design; software developments techniques and legacy systems; software evolution techniques; agile software and lean methods; software optimization and formal methods for software design; software maintenance; software security tools and techniques; formal techniques for software representation, software testing and validation; software reliability; model driven development (DVD), code centric to model centric software engineering; etc.
September 22-23	16th ACM/IEEE **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2022), Helsinki, Finland. Deadline for submissions: July 7, 2022 (Journal-First papers), August 13, 2022 (Industry Forum), September 19, 2022 (student volunteers).
September 27-29 (h)	19th **International Colloquium on Theoretical Aspects of Computing** (ICTAC'2022), Tbilisi, Georgia. 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; security; static analysis; software verification; software testing; runtime verification; model checking and theorem proving; applications and case studies; etc. Deadline for autumn school registration: July 31, 2022 (early), August 31, 2022 (late).

September 28-30 — 20th **International Conference on Software Engineering and Formal Methods** (SEFM'2022), Berlin, Germany. Topics include: software development methods; design principles; software testing, validation, and verification; applications and technology transfer; special topic "Software Engineering and Formal Methods for Intelligent and Learning Systems"; usage of formal methods in industrial applications, case studies, best practices, experience reports.

September 28-30 (h) — 22nd **International Conference on Runtime Verification** (RV'2022), Tbilisi, Georgia. Topics include: monitoring and analysis of runtime behaviour of software and hardware systems. Application areas of runtime verification include cyber-physical systems, autonomous systems, safety/mission critical systems, enterprise and systems software, cloud systems, reactive control systems, health management and diagnosis systems, and system security and privacy.

September 28-30 — 16th **International Conference on Verification and Evaluation of Computer and Communication Systems** (VECoS'2022), Tbilisi, Georgia. Topics include: analysis and communication systems, where functional and extra-functional properties are inter-related; cross-fertilization between various formal verification and evaluation approaches, methods and techniques, especially those developed for concurrent and distributed hardware/software systems.

October 03-06 (v) — 29th IEEE **Software Technology Conference** (STC'2022), Internet. Topics include: software engineering for emerging systems; software testing, testability, and assurance; cybersecurity and information assurance; agile software development; challenges and opportunities in SW & systems development processes; etc.

October 07-14 (h) — **Embedded Systems Week 2022** (ESWEEK'2022), Shanghai, China. Includes CASES'2022 (International Conference on Compilers, Architectures, and Synthesis for Embedded Systems), CODES+ISSS'2022 (International Conference on Hardware/Software Codesign and System Synthesis), EMSOFT'2022 (International Conference on Embedded Software). Deadline for submissions: July 1, 2022 (industry papers, Industry Challenge pitches) papers, Ph.D. Forum).

- October 07-14 (h) — **International Conference on Compilers, Architecture, and Synthesis for Embedded Systems** (CASES'2022). Topics include: latest advances in compilers and architectures for high-performance, low-power embedded systems; software security for embedded systems, IoT, and CPS; architecture, design, and compiler techniques for reliability, and aging; modeling, analysis, and optimization for timing and predictability; validation, verification, testing, and debugging of embedded software; etc.
- October 07-14 — **International Conference on Hardware/Software Codesign and System Synthesis** (CODES+ISSS'2022). Topics include: system-level design, hardware/software co-design, modeling, analysis, and implementation of modern Embedded Systems, Cyber-Physical Systems, and Internet-of-Things, from system-level specification and optimization to system synthesis of multi-processor hardware/software implementations.
- October 10-14 (h) — ACM SIGBED **International Conference on Embedded Software** (EMSOFT'2022). 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 integrate computation, networking, and physical dynamics.

October 10-14 — 37th IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2022), Oakland Center, Michigan, United States. Events include: ACM SIGAda's HILT workshop (High Integrity Language Technology) on Tools and Languages in support of a Rigorous Approach to Software Development.

♦ October 14 — ACM SIGAda **High Integrity Language Technology International Workshop on Supporting a Rigorous Approach to Software Development** (HILT'2022), Ann Arbor, Michigan, USA. Co-located with ASE'2022. Organized by ACM SIGAda, in cooperation with Ada-Europe. Topics include: practical use of High Integrity languages, technologies, and methodologies that enable expedited design and development of software-intensive systems; practical use of formal methods at industrial scale; IDE-support for formal methods; model-level analysis tools for

systems like SysML, AADL, Lustre, or Simulink; continuous integration and deployment based on advanced static analysis tools; safety-oriented programming language features; qualification of language tools for critical systems use; etc.

October 16-20 (h) — 17th **International Conference on Software Engineering Advances** (ICSEA'2022), Lisbon, Portugal. Topics include: trends and achievements; advances in fundamentals for software development; advanced mechanisms for software development; advanced design tools for developing software; software performance; software security, privacy, safeness; advances in software testing; specialized software advanced applications; open source software; agile and Lean approaches in software engineering; software deployment and maintenance; software engineering techniques, metrics, and formalisms; software economics, adoption, and education; etc.

October 25-28 (v) — 20th **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2022), Beijing, China. Topics include: theoretical and practical aspects of automated analysis, synthesis, and verification of hardware, software, and machine learning (ML) systems; specifications and correctness criteria for programs and systems decision procedures and solvers for verification and synthesis program analysis and software verification analysis and verification of parallel and concurrent systems analysis of cyber-physical systems analysis and verification of machine learning algorithms and systems formal models and methods for security and privacy testing and runtime analysis based on verification technology applications and case studies verification in industrial practice; etc.

November 10-11 (v) — 18th **International Conference on Formal Aspects of Component Software** (FACS'2022), Oslo, Norway. Topics include: applications of formal methods in all aspects of software components and services; formal methods, models, and languages for components and services, including formal aspects of concrete component-based systems, including real-time/safety-critical systems, hybrid and cyber physical systems, ...; tools supporting formal methods for components and services; case studies and experience reports over the above topics; etc. Deadline for submissions: July 18, 2022 (abstracts, papers).

November 14-18 — 30th ACM **Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering** (ESEC/FSE'2022), Singapore.

November 15-17 — 24th **International Symposium on Stabilization, Safety, and Security of Distributed Systems** (SSS'2022), Clermont-Ferrand, France. Topics include: concurrent and distributed computing (foundations, fault-tolerance, and security); distributed, concurrent, and fault-tolerant algorithms; synchronization protocols; formal methods, validation, verification, and synthesis; etc. Deadline for paper submissions: August 5, 2022 (2nd deadline).

December 05-07 — 29th **Static Analysis Symposium** (SAS'2022), Auckland, New Zealand. In conjunction with SPLASH'2022 Topics include: static analysis as fundamental tool for program verification, bug detection, compiler optimization, program understanding, and software maintenance.

☺ December 05-08 — 43rd IEEE **Real-Time Systems Symposium** (RTSS'2022), Houston, Texas, USA. Topics include: addressing some form of real-time requirements such as deadlines, response times or delay/latency. Deadline for submissions: September 12, 2022 (RTSS@Work demos, brief presentations).

☺ December 05-10 — ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2022), Auckland, New Zealand. Topics include: all aspects of software construction and delivery, at the intersection of programming, languages, and software engineering. Deadline for submissions: July 10, 2022 (Onward! papers), July 18, 2022 (Student Research Competition), August 1, 2022 (Doctoral Symposium), August 8, 2022 (GPE abstracts, SLE abstracts 2nd round), August 12, 2022 (GPE papers, SLE papers 2nd round), August 15, 2022 (posters), August 19, 2022 (SPLASH-E), September 1, 2022 (workshop papers), September 5, 2022 (Onward! Essays).

Dec 05-10 — 15th ACM SIGPLAN **International Conference on Software Language Engineering** (SLE'2022). Topics include: software language engineering rather than engineering a specific software language; software language design and implementation; software language validation (verification and formal methods for languages, testing techniques for languages, simulation techniques for languages); software language integration and composition; software language maintenance (software language reuse, language evolution, language families and variability,

language and software product lines); 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: August 8, 2022 (abstracts), August 12, 2022 (papers), October 11, 2022 (artifacts).

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

2023

January 16-18 18th **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2023), Toulouse, France. Topics include: software development for high performance parallel systems; tools for compilation, evaluation, optimization of high performance parallel systems (compiler support, tracing, and debugging for parallel architectures, ...); embedded real-time systems, mixed criticality system support, dependable systems, ...; software support for embedded architectures (tracing and real-time analysis of embedded applications, runtime software); etc.

March 06-10 (h) 25th **International Symposium on Formal Methods** (FM'2023), Lübeck, Germany. Topics include: development and application of formal methods in a wide range of domains including trustworthy AI, software, computer-based systems, systems-of-systems, cyber-physical systems, security, human-computer interaction, manufacturing, sustainability, energy, transport, smart cities, healthcare and biology; techniques, tools and experiences in interdisciplinary settings; experiences of applying formal methods in industrial settings; design and validation of formal method tools; formal methods in practice (industrial applications of formal methods, experience with formal methods in industry, tool usage reports, experiments with challenge problems); tools for formal methods (advances in automated verification, model checking, and testing with formal methods, tools integration, environments for formal methods, and experimental validation of tools); formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, and method integration); special FM 2023 session on "Formal methods meets AI" (focused on formal and rigorous modelling and analysis techniques to ensuring safety, robustness etc. (trustworthiness) of AI-based systems); etc. Deadline for submissions: July 1, 2022 (tutorials), September 4, 2022 (abstracts), September 11, 2022 (full papers), November 20, 2022 (artefacts).

April 22-27 26th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2023), Paris, France. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), TACAS (Tools and Algorithms for the Construction and Analysis of Systems). Deadline for submissions: August 30, 2022 (nominations EAPLS Best Dissertation Award), October 13, 2022 (papers), November 10, 2022 (TACAS artefact submissions), January 5, 2023 (ESOP, FASE, FoSSaCS artefact submissions).

♦ June 13-16 27th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC 2023 aka Ada-Europe 2023), Lisbon, Portugal. Sponsored by Ada-Europe.

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

HILT 2022 Workshop on *Supporting a Rigorous Approach to Software Development*

As part of. 37th IEEE/ACM International Conference on Automated Software Engineering, ASE'2022, October 14, 2022, Oakland Center, Michigan, United States

Sponsored by ACM SIGAda, in cooperation with Ada-Europe

This is the seventh in the HILT series of conferences and workshops focused on the use of High Integrity Language Technology to address challenging issues in the engineering of highly complex critical software systems. HILT 2022 will focus on the increasing synergies between formal methods (theorem provers, SAT, SMT, etc.), advanced static analysis (model checking, abstract interpretation), software design and modeling, and safety-oriented languages.

Keynote speakers:

Rustan Leino, Amazon Web Services

Recent experience with developing formally verified software

Senior Principal Applied Scientist in the Automated Reasoning Group at Amazon Web Services. Throughout his career, he has developed and applied tools for the formal verification of software. The most recent of these is the Dafny language and verifier, which has been used in projects and education for more than a decade. Leino is an ACM Fellow and a recipient of the CAV Award.

Niko Matsakis, Amazon Web Services

a-mir-formality: a formal model for the Rust language

Senior Principal Engineer at AWS and co-lead of the open source Rust language design team. He has worked on Rust since 2011, and led the design of its "secret sauce", the borrow checker. He has played a number of other roles in Rust over the years, such as being a member of the Rust core team, the lead of the Rust compiler team, and helping to launch the Rust Foundation.

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 rigorous software development.

HILT 2022 Schedule:

0830-0840 Opening, S. Tucker Taft and Jerome Hugues

0840-0940 Keynote#1: K. Rustan M. Leino, Senior Principal Engineer, Amazon Web Services.
"*Recent experience with developing formally verified software"*

0940-1130 Session #1: Formal methods and applications
1. Daniel Larraz (The University of Iowa) and Cesare Tinelli (The University of Iowa), *"Finding Locally Smallest Cut Sets using Max-SMT."*

[1010-1030 Coffee break]

2. Daniel Larraz (The University of Iowa), Arjun Viswanathan (The University of Iowa), Mickaël Laurent (Université de Paris) and Cesare Tinelli (The University of Iowa). *"Beyond model checking of idealized Lustre in Kind 2."*
3. Danielle Stewart (University of Minnesota) and John Hatcliff (Kansas State University). **"***An AADL Contract Language Supporting Integrated Model- and Code-Level Verification."*

1130-1230 Keynote #2: Niko Matsakis, Senior Principal Engineer, Amazon Web Services
"a-mir-formality: a formal model for the Rust language"

[1230-1330 Lunch]

1330-1530 Session #2: Language and Assurance
1. David Hardin (Collins Aerospace). *"Hardware/Software Co-Assurance for the Rust Programming Language Applied to Zero Trust Architecture Development."*
2. Claire Dross (AdaCore). *"Containers for Specification in SPARK."*
3. S. Tucker Taft (AdaCore). *"Rigorous Pattern Matching as a Language Feature."*

[1500-1530 Coffee break]

1530-1700 Session #3: Use Cases
1. Tabea Bordis (Karlsruhe Institute of Technology), Tobias Runge (Karlsruhe Institute of Technology), Alexander Kittelmann (Karlsruhe Institute of Technology) and Ina Schaefer (Karlsruhe Institute of Technology). *"Correctness-by-Construction: An Overview of the CorC Ecosystem"* (Short Abstract).
2. Laura Humphrey (Air Force Research Laboratory). **"***Basic Formal Verification of a Waypoint Manager for Unmanned Air Vehicles in SPARK."*
3. Howard Ausden (Leidos). *"Getting to 100% availability in a large C++ and Ada program."*

Organizing Committee

- Jerome Hugues, CMU/SEI (Co-Chair)
- Tucker Taft, AdaCore, Inc (Co-Chair)
- Dirk Craeynest, ACM SIGAda International Representative, KU Leuven
- Luis Miguel Pinho, Secretary-Treasurer, ACM SIGAda, Polytechnic Institute of Porto
- Alok Srivastava, Editor, ACM Ada Letters, SAIC

The 27th Ada-Europe International Conference on Reliable Software Technologies **(AEiC 2023)** will take place on **June 13-16, 2023**, in **Lisbon**, **Portugal**

Yes, it's true, AEiC will be returning to Lisbon!!!

If you were unable to enjoy the 2018 edition of AEiC, in Lisbon, this will be a great opportunity to visit the city while enjoying what the conference has to offer: keynotes, presentations, posters, vendor booths, and interactions with other participants.

If you attended the 2018 edition, then you know how nice it is downtown Lisbon, the taste of Portuguese food, the weather. And you will not want to miss the opportunity to return to Lisbon.

We are working to prepare a great event, with an interesting scientific and industrial program, and with enjoyable social, cultural and touristic opportunities.

Getting there is easy, with plenty of flights from all major cities in Europe, and from several cities in the US and Canada.

And when you land, you are already almost downtown!

Defining a Pattern Matching Language Feature for Ada

S. Tucker Taft
AdaCore; Lexington, MA, USA; email: taft@adacore.com
Stephen Baird
AdaCore; Mountain View, CA, USA; email: baird@adacore.com
Claire Dross
AdaCore; Paris, FR; email: dross@adacore.com

Abstract

Structural pattern-matching as a language feature has become more common in programming languages over the past decade. This talk will report on the work in progress to define such a feature for the Ada language, both from a language-design point of view, and from an implementation point of view.

Keywords: pattern-matching, language-design, Ada.

1 Introduction

Many programming languages now include a pattern-matching feature, often introduced with the keyword **match**, e.g. OCaml[1], Python[2], Haskell[3] (Haskell doesn't require any keyword -- every function is considered a pattern match). These are not primarily focused on string pattern matching, but more on structure pattern matching, where the matching starts from an object of some structured type, and the individual patterns select particular structural patterns for special handling.

These pattern matching features can be seen as a generalization of the **case** or **switch** statement available in most third-generation programming languages. But they typically include the ability to associate an identifier with some or all of the pattern, which is then usable inside the handler for the given pattern, knowing that that identifier refers to some part of the original object that satisfies the given part of the pattern.

In general, for a pattern matching language feature (including one as simple as a switch/case statement) three properties are considered important in any legal usage of the feature:

1. Complete/Exhaustive -- Every possibility is covered by some pattern.

   This is easily accomplished if the programmer includes a final *catch-all* pattern (e.g. others/default option), but it may be valuable in some cases to avoid such a *catch-all* and have the compiler complain unless the other more specific patterns cover all interesting cases.

2. Unambiguous -- There should be no two patterns where a given object could match both, unless the pattern that comes lexically second covers a strict superset of objects of the earlier pattern. The later, superset pattern is analogous to the **others/default** branch of a **case** or **switch** statement, which must come after all more specific patterns.

3. Nonredundant -- There are no patterns which are redundant, in that no object will *reach* that pattern, since it is fully covered by earlier patterns.

   This would be analogous to an **others/default** branch of a **case/switch** statement that is never reached, and could be considered misleading. Violating this property might be treated more as deserving a warning rather than an error, at least for a true *catch-all* pattern. For a pattern that is not a true *catch-all* but which is being used as a *fall-back* pattern for one or more earlier more specific patterns, if this *fall-back* is never reachable, then it might well be considered a *redundancy* error.

This talk will describe the work in progress to define a generalization of the Ada **case** statement which would provide a general, structural pattern-matching facility, and describe some of the language design issues and some of the interesting implementation challenges[4].

References

[1] X. Leroy, D. Doligez, A. Frisch, J. Garrigue, D. Rémy and J. Vouillon, “Chapter 9. The OCaml Language, Section 6. Patterns”, *The OCaml Manual,* https://ocaml.org/manual/patterns.html, retrieved 25-Feb-2022.

[2] Python Software Foundation, “8.6 The Match Statement”, *The Python Language Reference*, https://docs.python.org/3/reference/compound_stmts.html#the-match-statement, retrieved 25-Feb-2022.

[3] S. Marlow (ed.), “3.17 Pattern Matching”, *Haskell 2010 Language Report*, 2010. https://www.haskell.org/onlinereport/haskell2010/haskellch3.html, retrieved 25-Feb-2022.

[4] F. Le Fessant and L. Maranget, “Optimizing Pattern Matching”, *ACM SIGPLAN Notices* 36, 10.1145/507635.507641, 2001.

A Work-Stealing Scheduler for Ada 2022, in Ada

S Tucker Taft

AdaCore; Lexington, MA, USA.; email: taft@adacore.com

Abstract

Ada 2022 includes parallel programming features that use lightweight logical threads of control on top of the heavier-weight Ada tasks. This talk will report on the work in progress to implement a work-stealing scheduler for these lightweight threads, in Ada itself.

Keywords: parallel programming, work stealing, Ada 2022.

1 Introduction

Ada 2022 parallel programming features [1] rely on having multiple (lightweight) logical threads of control for a single Ada (heavyweight) task. This implies the need for a scheduler for such lightweight threads (LWTs). The OpenMP API [2] includes a scheduler which can be used as the basis for this LWT scheduler, as OpenMP has the notion of both lightweight threads, known (confusingly from an Ada perspective) as OpenMP "tasks") and heavier weight threads known simply as OpenMP "threads". An explicit goal of the design of Ada 2022 was to permit LWT schedulers such as the one provided by OpenMP to handle the scheduling for Ada 2022's "logical threads of control."

It is also possible to build an LWT scheduler directly in Ada, using Ada tasks as heavier weight *server* threads to provide the actual execution resources for the scheduler. This presentation will describe a work in progress to implement a high-performance lightweight thread scheduler in "pure" Ada, based on *work stealing*.

2 Work Stealing

Work stealing is widely recognized as an efficient approach to lightweight thread scheduling and has been adopted by many parallel-programming systems. Work-stealing became widely known as a result of work by Blumofe and Leisersen [3] at MIT in the context of the Cilk parallel programming language. Work stealing provides a nearly ideal combination of efficient load balancing across a set of cores (or kernel threads), while preserving locality of reference on a single core and separation of reference between distinct cores, which together minimize the unwanted cache effect known as *false sharing.*

The basic technique of work stealing is to have a separate, doubly-ended queue ("deque") of lightweight threads for each heavyweight *server* thread. In the context of Ada this means that we use multiple (anonymous) Ada tasks, each with its own deque, to service the various logical threads of control implicit in the use of the Ada 2022 parallel programming constructs of parallel loops and parallel blocks.

When a new light-weight thread (LWT) is spawned, the spawning server adds the LWT to the top of its own doubly-ended queue. When a server runs out of work, it pops the thread from the top of its deque. Hence, the LWTs are managed as a LIFO stack by a given server. Note that the LWTs at the *bottom* of this LIFO stack typically represent bigger jobs that have been languishing longer, if we are presuming new LWTs are spawned as part of a divide-and-conquer approach, with each new LWT spawned representing a shrinking fraction of the overall work to be done. This means that when a server runs out of LWTs on its own deque, it *steals* from some other server, and when it steals, it steals from the *bottom* of the stack of that server, so it gets an older and bigger job to do.

So this deque is effectively a LIFO stack on one end, used by a single server, and a FIFO queue at the other end, used competitively by multiple servers.

There is a "classic" implementation of the work-stealing deque by Chase and Lev [4]. We implemented this algorithm using the Ada 2022 Atomic_Operations.Exchange package, and then refined it to take advantage of Ada's modular types. This talk will report on some of the specific challenges in achieving high performance, and the results achieved in the implementation of various Ada 2022 parallel programming features, in comparison with the use of the OpenMP scheduler.

References

[1] ISO Wg9, *The Ada Reference Manual,* Clause 9 Tasks and Synchronization, 2022. http://www.ada-auth.org/standards/2xrm/html/RM-9.html, retrieved 26-Feb-2022.

[2] OpenMP ARB, *OpenMP Application Programming Interfaces*, Section 1.3 Execution Model, 2021. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf, retrieved 26-Feb-2022.

[3] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing", *Journal of the ACM,* 46, 5 pp. 720–748, DOI: https://doi.org/10.1145/324133.324234, 1999

[4] D. Chase and Y. Lev, "Dynamic circular work-stealing deque", *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures* (SPAA '05), ACM, New York, NY, USA, 21-28, 2005, DOI: http://dx.doi.org/10.1145/1073970.1073974, https://www.dre.vanderbilt.edu/~schmidt/PDF/work-stealing-dequeue.pdf, retrieved 26-Feb-2022.

Resilience-Aware Mixed-Criticality DAG Scheduling on Multi-cores for Autonomous Systems

Jie Zou, Xiaotian Dai, John A. McDermid
Department of Computer Science, University of York, UK; email: {first.last}@york.ac.uk

Abstract

Fully- and semi-autonomous systems are complex and safety-critical with strict timing and resource constraints, and have a deep processing pipeline with strong dependencies between different functions. Furthermore, tasks with different criticalities share the same hardware, and the scheduling strategy has to guarantee high criticality tasks' execution irrespective of interference from low criticality tasks whilst respecting the precedence constraints among tasks. Most static scheduling work considering task dependencies does not take into account the survivability of low criticality tasks, instead assuming that all low criticality tasks should be suspended or discarded after a mode change. Consequently, the schedules for high and low modes are different, so that more effort is needed to check the safety of schedules during mode change and with a potential increase in the migration cost as tasks may be executed on a different core after a mode change. This work proposes a novel mixed-criticality DAG-based multi-core static scheduling method considering low criticality tasks' survivability and precedence constraints between tasks with different criticalities. This produces a consistent schedule for different system modes enabling task-level mode change and improving the resilience of the system. Furthermore, the utilisation of computational resources is also improved by avoiding discarding low tasks.

Keywords: mixed-criticality systems, DAG scheduling, resilient autonomous systems

1 Background and Related Work

The development of advanced driver-assistance systems (ADAS) and semi-autonomous systems, which comprise increasing number of tasks with different criticality levels, has been intensively studied in recent years. Mixed-criticality systems (MCSs) provide the opportunity to integrate tasks with different criticality requirements onto a shared computational platform. At the same time, there are increasingly stringent predictability requirements to support safety certification. Baruah and Fohler remarked that static scheduling is well supported in the industry because it is regarded as giving "complete determinism" and is easy to certify [1]. Thus, static scheduling is adopted as the foundation for this work.

In this work, we use DAGs to describe the functions of systems. DAGs have been widely used as they allow intuitive identification of task dependencies and parallelizable tasks. Recently, some state-of-the-art strategies proposed generating static schedules based on mixed-criticality DAGs, e.g. [2], [3], [4], [5] and [6]. However, most existing static scheduling methods use different schedules for systems in different modes (HI and LO modes) without considering the survivability of LO criticality tasks (in HI mode).

Nevertheless, as Bletsas et al. observed in [7], some tasks considered as LO criticality might still contain mission-critical functions and are vital for the correct and efficient operation of the system. Thus, Burns et al. [8] emphasise the importance of robustness and resilience of task scheduling for mixed-criticality systems. Furthermore, as Adjur et al. identified [9], the overrun of one specific HI criticality task does not imply that all HI tasks simultaneously exhibit their longest Worst-Case Execution Time (WCET). Propelled by this motivation, we propose what we believe to be the first approach to generating one consistent static schedule considering the survivability of LO criticality tasks to improve the resilience of the system. In this paper, we briefly introduce the idea and the formulation of the proposed scheduling with a case study.

2 Formulation

This section introduces the proposed resilience-aware mixed-criticality multi-core DAG scheduling method, where the schedule formulation is done in two steps.

2.1 Mixed-Criticality Task Model

We assume a dual-criticality system considering criticality-dependent WCET estimation (i.e., for HI critical tasks $C(LO) \leq C(HI)$). A task τ_i can be defined by the tuples $(T_i, D_i, C_i(HI), C_i(LO), L_i)$, where T_i, D_i represent the period and deadline, respectively ($T_i = D_i$); L_i is the criticality level; and C_i denotes the execution time. In this work, we introduce the idea of minimum operation, i.e., a LO task will always keep a simple data refresh operation; this is to ensure the timeliness of data and the operation is assumed to consume one time unit. Thus, if L_i is LO, then $C_i(HI)$ is one. If the schedule considers the parallelism of LO tasks, as Figure 1 shows, the overrun of HI_1 task only impacts on $LO_{1.1}$. According to our proposed strategy, $LO_{1.1}$ will keep a minimum data refresh operation, and because of the

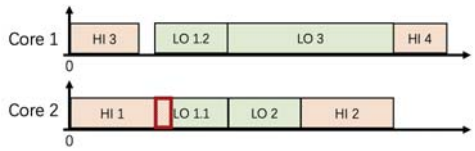

Figure 1: HI mode schedule example with degraded LO criticality task

existence of $LO_{1.2}$, task LO_1 is regarded as surviving with degraded performance. Furthermore, it is worth noting that $LO_{1.2}$, LO_2 and LO_3 will never be discarded because their execution will never be affected. Evidently, an appropriate static schedule can ease the task-level mode change because for each HI task, the LO tasks which could be impacted by its overrun, are not difficult to be identified once the schedule is known.

2.2 Consistent Mixed-Criticality DAGs (CMC-DAG) Scheduling

This subsection defines our proposed new *Consistent Mixed-Criticality DAGs* (CMC-DAG) scheduling method. For DAG scheduling, the execution of one task can be started only when all its predecessors finish their execution. When calculating the schedule using backpropagation, a task can only be treated when the time slots for all of its successors have been identified. Therefore, the definition of each task should be extended to $\tau_i := (T_i, D_i, C_i(HI), C_i(LO), Li, Suc(\tau_i), Pred(\tau_i))$, where $Suc(\tau_i)$ and $Pred(\tau_i)$ represent the set of successors and predecessors of τ_i, respectively.

Before the schedule calculation, the worst-case finishing time (f_i) of each node needs to be calculated following Equation (1), where $CP(\tau_i)$ denotes the critical path of task τ_i; G_i is DAG i; N_{G_i} represents the number of times the DAG is released (starting from one).

$$f(\tau_i) = (N_{G_i} - 1) \times T(G_i) + CP(\tau_i) \tag{1}$$

To provide more flexibility for LO task allocation, we adopt the idea proposed in [3], where the schedule calculation is started from the last time point of the hyperperiod and the last layer of DAGs in the system (i.e., it starts from the last task of the system back to the source node). Thus, all HI critical tasks can be executed as late as possible. Their scheduled start time will be fixed and kept the same in both system modes. Similar to the state-of-the-art methods, the schedule calculation also consists of two steps.

2.2.1 Schedule calculation in HI mode (Step 1)

The schedule should be generated based on the HI mode task behaviour to guarantee the execution of HI tasks. The existence of LO criticality tasks can assist in anchoring the target schedule region of LO tasks in LO mode, which guarantees the satisfaction of precedence constraints among tasks with different criticalities in both system modes. The anchor time point is recorded as the latest starting time of minimum operation for each LO task. Besides, it can assist in grouping the LO tasks to specific a HI task to realise task-level mode change.

Starting from the bottom layer of DAGs, the priority value $P(\tau_i)$ of each task from the same layer will be calculated according to Equation (2). Then the task priority will be assigned according to the priority value sorted in increasing order. Moreover, the relative order among LO critical tasks should be adapted to increase the probability that the allocation of the regular (non-parallelizable) LO task will be prior to the sub-task of the segmented LO task. In this work, time slot allocation is in the backpropagation direction; the first scheduled regular LO task would be executed following the sub-task of the segmented LO task after schedule reversing. Then, the overrun of the previous HI task can impact the sub-task first, and the normal one could be protected.

$$P(\tau_i) = D_i - f(\tau_i) \tag{2}$$

Based on the adapted list, the task at the beginning of the queue will be selected and allocated to the core with maximum remaining utilization U_r. Then, the utilization value of that core should be updated according to Equation (3), where $S(\tau_i)$ denotes the start time of the allocated task (relative to this hyperperiod). $C(\tau_i)$ represents the execution time of the allocated task. For HI tasks, $C(\tau_i) = C(HI)$ and for LO tasks, $C(\tau_i) = C(LO)$ because we need to be aware of the execution behaviour of each LO task in LO mode. In this step, the start time of each HI task is fixed. It is vital to note the precedence constraints during start time allocation. With the help of anchor time points of LO tasks and the fixed start time of HI critical tasks, the LO tasks which can be impacted by each HI task can be grouped.

$$U_r \leftarrow min\left\{\frac{S(\tau_i)}{T_{hyper}}, U_r - \frac{C(\tau_i)}{T_{hyper}}\right\} \tag{3}$$

2.2.2 Consistent schedule generation (Step 2)

Based on the schedule calculated in step 1, all tasks are performed with LO mode behaviour in this step. The execution time of HI tasks is shortened, and more time is freed to schedule LO tasks. The allocation of LO tasks is group-based. In each HI task group, the related LO tasks are fixed, and the last time point for task allocation is the start time of the next HI task on the same core. We also adopt the back-propagation allocation method, working from the end to the beginning time point; thus, the regular type of LO critical task from a specific HI task group would be scheduled first, and the possible end time is the start time of the following HI task. After the task is allocated, its start time is regarded as the end time of the next scheduled LO task. After identifying the possible end time of LO tasks, the upper bound of starting time is defined by the end time of the owner (HI task) of the specific group or the latest finish time of all its predecessors. If the slack time from the same core is not sufficient to support the execution of the LO task, its execution should be segmented. The remaining execution can be allocated to other cores, and regarded as a new task, which would be fed into the waiting queue.

After all possible time slots have been allocated, we will go back to check the schedulability of tasks in the waiting

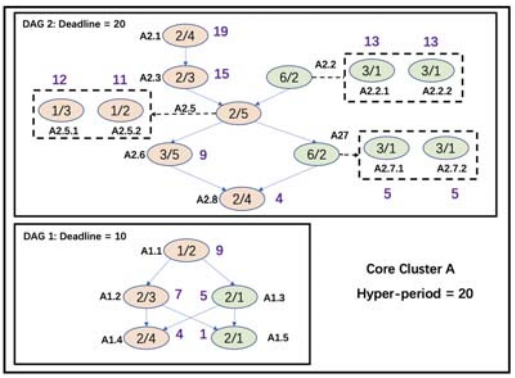

Figure 2: Mixed-criticality DAGs example (nodes in green: LO-tasks; nodes in red: HI-tasks)

queue. First of all, the queue should be sorted according to the priority value determined by Equation (2) in increasing order and start from the one with the minimum value. We only need to search for free time slots of each core in the LO mode schedule for the remaining execution of one specific LO criticality task. However, for tasks that are fully unallocated in the waiting queue, the slack time of each core under HI and LO mode should be checked because, in LO mode, there is no free space to allocate it on the expected core, even for the minimum operation. Considering the consistency of static schedules under all system modes, the anchor time point should also be migrated to other cores. Thus, The overlapped free time can be used to allocate at least the one-time unit minimum operation, where possible. The remaining part would be scheduled in the same way determined for the segmented task in the waiting queue. If the waiting tasks cannot be scheduled successfully, the system is regarded as unschedulable.

In summary, in this work, a consistent schedule enables task-level mode change without any other effort for system mode recovery. Furthermore, the survivability of LO tasks and the resilience of systems are considerably improved, and the computational resources can be used with high efficiency.

3 Case Study

A case study shown in Figure 2 is used to demonstrate the schedule calculation procedures and highlights the advantage of the proposed method. The example system consists of two functions, represented by mixed-criticality DAGs with different periods and mapped to core cluster A. To make the case study more practical, task $A2.5$ is regarded as a parallelizable HI task, segmented into $A2.5.1$ and $A2.5.2$, and deployed on different cores to accelerate the execution. The LO tasks $A2.2$ and $A2.3$ can be segmented, and the missing part of their execution only implies performance degradation. First of all, we adopt the method introduced in [10] to determine the number of cores used for this example. Because of space limitations, the procedure will be left to a later paper. In this case study, the utilization of the system in HI mode equals 2.35, and for LO mode, the utilization equals 2.05. Thus, the minimum number of cores is three.

Then, within the hyper-period 20, DAG 1 is executed two times with period of 10. The finish time of each node in

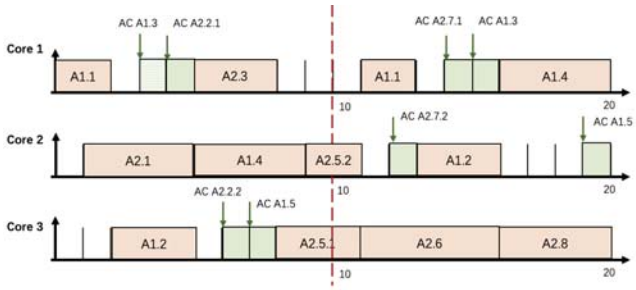

Figure 3: The schedule generated by step 1

HI mode can be calculated following Equation (1), and is marked in purple. Following the procedure introduced in step 1, we calculate the schedule in the backpropagation direction (from time point 20 to 0), starting from the first layer of the DAG which consists of three tasks $A1.4, A1.5, A2.8$. The priority values obtained following Equation (2) are 6, 9 and 16, respectively. Thus, task $A1.4$ is selected first and is allocated to core 1. The start time is fixed as 16, and the remaining utilization of core 1 will be reduced to 0.8. Then, considering $A1.5$, we use worst-fit to balance the workload of different cores and the remaining utilization of core 2 is calculated based on the execution time of the allocated tasks in LO mode (i.e., 2 time units for $A1.5$) and reduced to 0.9. The anchor time point of $A1.5$ is set to 19. Finally, $A2.8$ is allocated to core 3 and the utilization is updated to 0.8. Then, we move to layer 2 in the DAG, which comprises five tasks ($A1.2, A1.3, A2.6, A2.7.1, A2.7.2$). The priorities are 3, 5, 11, 15 and 15, respectively. The order of selection is (A1.2, A1.3, A2.6, A2.7.1, A2.7.2). When checking the relative order of LO tasks, the regular $A1.3$ is selected before the segmented task $A2.7$. Therefore, we do not need to swap the relative order. In this round, core 2 has the maximum remaining utilization with 0.9, so $A1.2$ will be allocated to core 2. Then we need to check the earliest start time of its successors (i.e., $A1.4$ and $A1.5$), and the time point of $A1.4$ with 16 is selected. Thus the start time of $A1.2$ on core 2 is set to 13, as calculated by $S_{\tau_i}(HI) = min\{S_{Suc(\tau_i)}(HI)\} - C_{\tau_i}(HI)$, and the remaining utilization reduces to 0.65. $S_{Suc(\tau_i)}$ denotes the start time of τ_i's successors. Following the same rules, the static schedule in HI mode is calculated based on a non-preemptive strategy. When it comes to the last layer in the DAG ($A1.1$), the remaining utilization of each core is 0, 0, and -0.05, respectively. Thus, we need to check the slack time of each core based on the existing schedule. Considering the precedence constraints, $A1.1$ should finish its execution before the start time of $A1.2$. $A1.1$ does not have any predecessors thus the slack time searching duration is [0, 2]. On core 1, there is a slack time slot, and the start time of $A1.1$ will be set as the start time of the slack, which equals 0, here. Finally, the schedule from step 1 can be generated as shown in Figure 3.

Following step 2 for consistent schedule design, the LO task group of $A1.1$ contains $A1.3, A2.2.1$, which implies that before the start time of the next HI task $A2.3$ allocated to the same core, the schedule of all tasks from this group should be allocated. Starting from the regular task, which should be

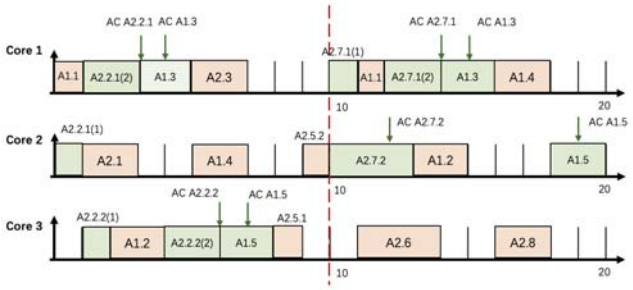

Figure 4: The consistent schedule from step 2

selected first, the anchor time point of $A1.3$, $A2.2.1$ will be changed. In LO mode, the execution time of $A1.1$ is shortened to one time unit, and the start time of $A1.3$ is set to 3. The remaining slack time from the same core is 2 and not sufficient to support the execution of $A2.2.1$ thus, it will be segmented and the portion allocated to core 1 is marked as $A2.2.1(2)$ and the remaining execution is regarded as a new task with one-time unit and fed into the waiting queue. For the second released task $A1.1$, the time duration for $A2.2.1$ starts from the latest end time of its predecessors (i.e., 10 from $A2.5.2$) with $C(LO)$ to the earliest start time of its successors or the following task on the same core (i.e., 14 from $A1.3$). Then we can find that its execution should be segmented into $A2.7.1(1)$ (from 10 to 11) and $A2.7.1(2)$ (from 12 to 14). After finishing all possible allocation, returning to the waiting queue, there is only one waiting task $A2.2.1$, in this example. When facing multiple waiting tasks, the one with the lowest priority value will be selected first. The search time duration for the remaining task $A2.2.1$ is bounded by 0 and the start time of $A2.2.1(2)$. On core 2, one free time slot can be found, i.e., [0, 1] and the search would be stopped and the portion marked as $A2.2.1(1)$. Finally, a consistent schedule can be generated as shown in Figure 4. Though the migration of tasks during run time cannot be fully avoided, comparing with the schedule generated by preemption according to the priority of each task in paper [10], the number of migrations can be significantly reduced.

Based on the consistent schedule, it is not difficult to find that the overrun of each HI task can only interfere with the execution of the LO tasks in its group, and there exist LO tasks that will never be discarded, such as $A1.5$. Besides, task $A1.3$ will be degraded to minimum operation only if $A2.2.1$ can not finish its data refresh before its anchor time point. The overrun of $A1.1$ only leads to the performance degradation of $A2.2$ because of the loss of $A2.2.1$. Suppose the minimum operation is not started before its anchor point. All LO critical tasks should be discarded with only the minimum operation kept. For example, after the overrun of $A1.1$, ideally, the minimum operation can be finished before instant 3. If not, that will trigger the discarding of $A1.3$. Time slots [3,4] and [4,5] can guarantee the minimum execution of $A2.2$ and $A1.3$, respectively.

4 Conclusion

In this work, a novel mixed-criticality, multi-core DAG scheduling strategy is proposed. Instead of generating two different schedules for the different system modes and discarding all LO critical tasks in HI mode, our strategy generates one consistent schedule considering the survivability of LO tasks to reduce the complexity of task-level mode change, which can accelerate the recovery of specific impacted LO tasks. The existence of LO criticality tasks also improves the efficiency of computational resource utilisation. Whilst we have illustrated the approach using an example and illustrating the schedules graphically. However, the method will be applied to more realistic examples — as we are doing with a mobile delivery robot.

References

[1] S. Baruah and G. Fohler, "Certification-cognizant time-triggered scheduling of mixed-criticality systems," in *2011 IEEE 32nd Real-Time Systems Symposium*, pp. 3–12, IEEE, 2011.

[2] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic dag tasks," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, pp. 227–236, IEEE, 2016.

[3] R. Medina, E. Borde, and L. Pautet, "Directed acyclic graph scheduling for mixed-criticality systems," in *Ada-Europe International Conference on Reliable Software Technologies*, pp. 217–232, Springer, 2017.

[4] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real-time systems*, vol. 53, no. 5, pp. 760–811, 2017.

[5] R. M. Pathan, "Improving the schedulability and quality of service for federated scheduling of parallel mixed-criticality tasks on multiprocessors," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[6] R. Medina, E. Borde, and L. Pautet, "Generalized mixed-criticality static scheduling for periodic directed acyclic graphs on multi-core processors," *IEEE Transactions on Computers*, vol. 70, no. 3, pp. 457–470, 2020.

[7] K. Bletsas, M. A. Awan, P. F. Souto, B. Akesson, A. Burns, and E. Tovar, "Decoupling criticality and importance in mixed-criticality scheduling," in *Workshop on Mixed Criticality*, pp. 25–32, York, 2018.

[8] A. Burns, R. I. Davis, S. Baruah, and I. Bate, "Robust mixed-criticality systems," *IEEE Transactions on Computers*, vol. 67, no. 10, pp. 1478–1491, 2018.

[9] J. Boudjadar, S. Ramanathan, A. Easwaran, and U. Nyman, "Combining task-level and system-level scheduling modes for mixed criticality systems," in *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 1–10, IEEE, 2019.

[10] R. Medina, E. Borde, and L. Pautet, "Scheduling multi-periodic mixed-criticality dags on multi-core architectures," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, pp. 254–264, IEEE, 2018.

Artificial Neural Networks for Real-Time Data Quality Assurance

Inês Sousa, António Casimiro, José Cecílio

Faculty of Sciences, University of Lisbon, Lisbon, Portugal; email: fc51588@alunos.fc.ul.pt, casim@ciencias.ulisboa.pt, jmcecilio@ciencias.ulisboa.pt

Abstract

Wireless Sensor Networks used in aquatic environments for continuous monitoring are typically subject to physical or environmental factors that create anomalies in collected data. A possible approach to identify and correct these anomalies, hence to improve the quality of data, is to use artificial neural networks, as done by the previously proposed ANNODE (Artificial Neural Network-based Outlier Detection) framework [1].

In this paper we propose ANNODE+, which extends the ANNODE framework by detecting missing data in addition to outliers. We also describe the design and implementation of ANNODE+, implemented in Python to exploit readily available machine learning (ML) tools and libraries, also allowing online processing of incoming measurements. To evaluate the ANNODE+ capabilities, we used a dataset from a sensor deployment in Seixal's bay, Portugal. This dataset includes measurements of water level, temperature and salinity. We observed that our implementation of ANNODE+ performed as intended, being able to detect injected anomalies and successfully correcting them.

Keywords: Neural networks, Environmental monitoring, Sensor networks, Forecasting, Data quality

1 Introduction

Nowadays, maintaining good water quality is important for the aquatic fauna and flora and our life quality. It has become a scarce resource, so it is crucial to monitor it. The Internet of Things (IoT) and Wireless Sensor Networks (WSNs) play an important role to monitor and inspect its quality. WSNs are networks with dedicated sensors that detect specific phenomenons or events. WSNs have been used to remotely monitor many different aquatic environments such as rivers, coasts, lakes and bays [1, 2].

Given that WSNs and their sensors are exposed to physical or environmental factors that often create anomalies in collected data, existing solutions can benefit from platforms for detecting erroneous data or data omissions, to provide the required reliability.

In this work, we propose the ANNODE+ framework, an artificial neural network-based framework for online data quality assurance. Taking inspiration from ANNODE, an outlier detection framework based on Artificial Neural Networks (ANNs) previously proposed by Jesus et al. [1], we report on the on-going design and implementation of a new, more generic and extended framework, usable in multiple settings. With the support of ANNs, the framework considers incoming measurements as time series (e.g., temperature values over time), predicting future values in the series. Each received measurement goes through a set of blocks to determine if it is an outlier, to estimate its quality, and, if considered an outlier, to replace it by a corrected measurement. Our framework was designed for online processing of incoming sensor measurements, and implemented with real-time concerns in mind, to reduce the time taken to process each incoming measurement and avoid arbitrarily large processing times. It offers capabilities to deal with a single sensor (data source) or multiple sensors providing correlated measurements. In fact, the ability to detect outliers can be significantly improved when correlated data sources are available. Measurements can be correlated if different variables have an impact within one another, e.g, salinity levels can change temperature levels. If more than one data source is available, some events can be explained as incidents. For instance, if it is detected a change in water levels, this change will also be detected in other sensors. However, if an event is detected by only one data source, it is most likely that that event is an anomaly.

2 Related Work

There have been many investigations and many different projects using WSNs. However, there is little work on the detection and correction of anomalies in sensor data, which can be frequent when considering deployments affected by harsh environmental conditions. We reviewed previous work done by different authors and the current state-of-the-art in the context of this work for each topic.

Ensuring that sensor data is of good quality, is of great importance to applications that rely on these types of data [3]. As discussed in the survey from Hui Yie Teh et al. [4], artificial intelligence (AI) and machine learning (ML) solutions are now commonly used to ensure data quality despite sensor and network failures. In [4], Hui Yie Teh et al. made a literature search for keywords related to these topics. Table 1 shows the most frequent types of sensor data errors mentioned in papers and the number of times they were mentioned.

From these values, it is clear that outlier and missing data errors are those that gather comparatively more attention in

Table 1: Most common types of errors in sensor data [4] and number of papers mentioning them.

Type of error	Total	Type of error	Total
Outliers	32	Noise	8
Missing Data	16	Constant value	7
Bias	12	Uncertainty	6
Drift	12	Stuck-at-zero	6

the literature, perhaps because they are prominent in real deployments. These are the two types of errors that we address with ANNODE+.

As previously stated, ML methods are commonly used to detect anomalies in sensor data. Several different methods, including supervised and unsupervised ones, can be used to detect anomalies based on past data. In [4], an analysis of the most common ML methods for error detection was also done, being the three most prominent ones the Principal Component Analysis (PCA), Artificial Neural Networks (ANNs) and Ensemble Classifiers. Our framework uses ANNs for this process, more specifically MultiLayer Perceptron Neural Networks (MLP).

In addition to detecting outliers, ANNODE+ also detects missing data. This is done only for periodic data, assuming that the period is known.

In [5], a similar experience was made in Aveiro, Portugal. The authors used and adapted a custom-deployment based forecasting platform to the Portuguese coast. This allowed to create numerical models to provide forecasts of sea level variations, currents, temperatures, etc. However, they also recognize that further research and solutions to deal with data errors are needed when considering sensor deployments in harsh environments. Furthermore, they refer to the possibility of exploiting existing temporal and spatial correlations in sensor data.

To understand the relevance of correlations between data from different sensors, several ML methods were considered in [6] to forecast long-term and short-term water demand when considering variables such as rain, hour of the day and air temperature. It was possible to conclude that using multiple correlated variables helps improving the accuracy of forecasts, with some variables having more impact on the achieved results.

In summary, the ANNODE+ framework is designed to detect and correct the most common sensor data errors. Additionally, the framework includes mechanisms, based on timers, to deal with missing data and exploit data correlations that may help with the predictive model and the detection of anomalies.

3 ANNODE+ Architecture

The framework's architecture is illustrated in figures 1 and 2. It is composed of two blocks: the training and execution blocks. The training block corresponds to an offline execution for models' training. This training step is supported by a dataset containing sufficient information to represent all the main characteristics and dynamics of the variable being modelled (e.g., represent the seasonality present in the real phenomenon). The user must prepare a configuration file with all the training requirements, such as the number of sensors, their characteristics (e.g., sampling period), and data characteristics (e.g., representative period).

The characteristics of the MultiLayer Perceptron (MLP) neural networks trained in the framework are the same as those proposed in the ANNODE framework, which are described in [1], and consist of two hidden layers with 20 neurons in the first layer and 15 in the second, using a hyperbolic tangent sigmoid (tansig) as the activation function.

Models are trained using datasets that must be provided by the user, which must have been previously collected and must include only data considered correct. Annotating or cleaning training data is a typical requirement when using ML methods. When data from multiple correlated sensors is available, several models are created, corresponding to different combinations of sensors. In fact, to predict the next measurement of a sensor, it is possible to use a model that was trained using only data from that sensor (exploiting temporal correlation), or using a model combining data from multiple neighbor sensors (exploiting spatial correlation).

After training the ANN models, the framework is ready to run (online execution). The Execution block follows a multi-service implementation approach. Currently, there are three primary services: Communication, Omission detection and Processing. The Communication service is responsible for receiving sensor data, for identifying its source sensor, and for inserting these data in that sensor measurements queue, which is shared with the Omission detection service.

The Omission detection service is a multi-thread service with a timer thread for each sensor from which periodic data is to be received. Considering that each sensor s sends a new measurement with period P_s, a corresponding timer is set to expire after $P_s + \delta$, where δ corresponds to the jitter assumed for the measurement reception instant. If no measurement from that sensor is received before $P_s + \delta$, an omission is detected and a special value (like a NaN) is inserted in the sensor measurements queue. By considering the channel jitter, the probability of wrongly detecting an omission is reduced to the probability of considering a wrong (too small) jitter.

Lastly, there is the Processing service. This service is triggered by new incoming measurements, stored in each sensor's measurements queue. However, actual processing only starts after a certain number of measurements have been received, covering an entire representative period of the physical process being monitored (e.g., 12 hours for sea water level). When this condition is fulfilled, actual processing is executed for every new incoming measurement. Firstly, measurements are temporally aligned, to match the alignments used during model training. Then, input vectors are built from the stored and aligned measurements, to be fed to the relevant ANN prediction models. If correlated sensors are available, then the following models (which have been previously trained) are used to generate forecasts:

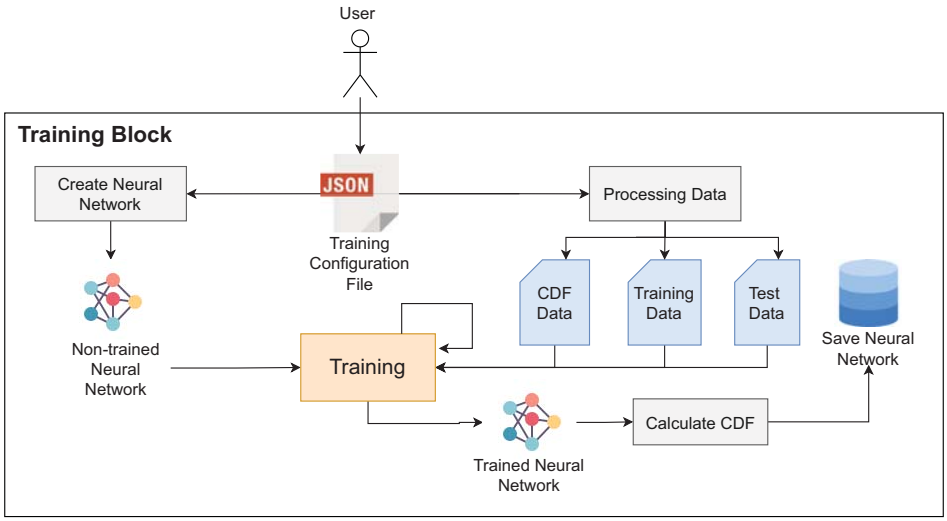

Figure 1: ANNODE+ Architecture - Training Block

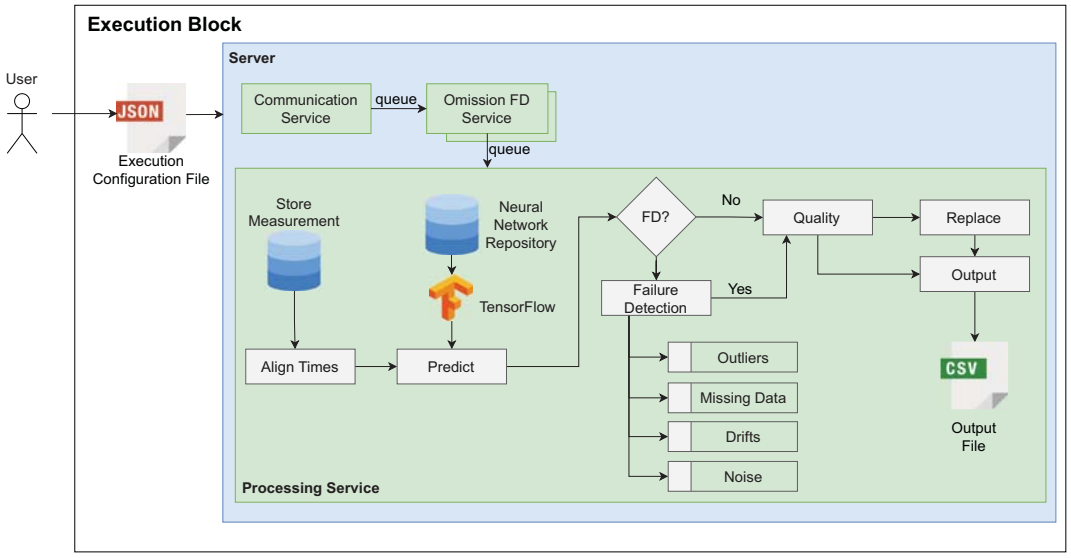

Figure 2: ANNODE+ Architecture - Execution Block

1. A model that only uses data from the target sensor whose measurement is being processed (temporal correlation);
2. A model that uses data from the target sensor and its neighbours (temporal and spatial correlation);
3. A model that uses data only from the target sensor neighbours (spatial correlation only).

The benefit of using these multiple forecasts is to be able to distinguish real environmental events (even if they look like outliers) from real outliers (only affecting the target sensor, but not the neighbor ones). It is then possible to check the correctness of the received measurement, comparing it with the generated forecasts. In fact, given that sensor data can be affected by different kinds of errors, it is at this point of the processing chain that it is possible to determine if some failure may have happened, leading to those different errors. While ANNODE+ is only detecting outliers, it may be extended in future work with new failure detection blocks to detect data drifts and noise. When some failure is detected (either missing data or outlier), then the received measurement (or the special NaN value) is replaced by an average of the calculated forecasts.

In addition to detecting failures, it is also possible to calculate the quality of the received measurement. When the difference between the received measurement and the forecasts provided by the models is small, then the quality is high.

4 Results

To evaluate our ANNODE+ implementation, we used a dataset with temperature measurements collected from a sensor in the Seixal Bay, in Portugal. We divided this dataset in two parts, containing data from different temporal periods. One part was used to train a single model (in this initial evaluation we only considered one model, exploiting temporal correlations) and the other was used to test the framework. Training a single model took about one full day on our hardware with 16GB of RAM and an AMD Ryzen 5 3500X CPU. Given our objective of checking the ability of the framework

to detect outliers and missing values, we randomly injected these anomalies in the second part of the dataset, by changing some measurements and by removing some of them from the temporal series.

To emulate a real online usage of the framework, we built a framework client that plays the role of a sensor and sends a new measurement (taken from the dataset) to the framework with a period of 500ms (except when injecting an omission).

We observed very positive results, with the framework replacing the injected outliers with their respective predictions. As stated before, when training is done with a sufficiently large and representative dataset, the obtained predictions are very accurate. We were also able to detect omissions, as expected, and correctly replacing the missing measurements with the values predicted by the ANN model.

Given the need to execute the full processing chain before the arrival of a new measurement, we also measured the response time of the framework, from the moment a new measurement arrives until it is fully processed, to assess the achievable performance and potential capacity of the framework in handling incoming data. The ability to ensure a bounded execution time necessarily depends on the underlying execution environment (namely the operating system being used), but having an idea of the time needed to process a single measurement is already important to make sure that the framework is not overloaded. In our experiments we also measured the CPU load, and we considered the execution of the framework with and without background load. The background load was created by running the framework for model training.

Table 2: Run-time cost of the algorithm for one measurement.

Without load	Total	With load	Total
Max time (ms)	165	Max time (ms)	153
Avg time (ms)	52	Avg time (ms)	54
Best time (ms)	46	Best time (ms)	49
Avg CPU load	11%	Avg CPU load	42%

Table 2 provides the collected performance measurements. The results show that in this case it would be possible to fully process each new measurement in about 50ms (in average), even considering a loaded CPU. For most applications performing environmental monitoring, this kind of performance is sufficiently good to allow the framework to be used for online processing of incoming measurements, even if measurements from several sensors have to be processed.

5 Conclusion and Future Work

The preliminary results show very positive outcomes. However, we aim to complete and test the implementation of ANN-ODE+ to handle more than one data source, and improve the calculation of the measurement quality using correlations between sensors. We also plan on improving the training block by automating some parts of the training process.

Moving away from applications in the environmental monitoring area, we plan to use the framework in a use case of arc detection in DC distribution cabinets, in the context of the VEDLIoT EU project (https://vedliot.eu). In this case there is a stream of sensor data being continually produced and sent in batches to the framework, while timing requirements for the detection of arcs (which will create data that will look like outlier in relation to normal data) are very stringent, in the order of a few milliseconds. A different execution platform will be required, with more resources and making it possible to satisfy timeliness requirements.

Acknowledgments

This work was supported by FCT through funding of the AQUAMON project (ref. PTDC/CCI-COM/30142/2017) and the LASIGE Research Unit (ref. UIDB/00408/2020 and ref. UIDP/00408/2020), and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 957197 (VEDLIoT project).

References

[1] G. Jesus, A. Casimiro, and A. Oliveira, "Using machine learning for dependable outlier detection in environmental monitoring systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 5, jul 2021.

[2] B. O'Flyrm, R. Martinez, J. Cleary, C. Slater, F. Regan, D. Diamond, and H. Murphy, "Smartcoast: A wireless sensor network for water quality monitoring," pp. 815 – 816, 11 2007.

[3] G. Jesus, A. Casimiro, and A. Oliveira, "A survey on data quality for dependable monitoring in wireless sensor networks," *Sensors*, vol. 17, no. 9, p. 2010, 2017.

[4] H. Teh, A. Kempa-Liehr, and K. Wang, "Sensor data quality: a systematic review," *Journal of Big Data*, vol. 7, 02 2020.

[5] J. Gomes, M. Rodrigues, A. Azevedo, G. Jesus, J. Rogeiro, and A. Oliveira, "Managing a coastal sensors network in a nowcast-forecast information system," in *2013 Eighth International Conference on Broadband and Wireless Computing, Communication and Applications*, pp. 518–523, IEEE, 2013.

[6] B. Brentan, G. Meirelles, M. Herrera, E. Luvizotto Jr, and J. Izquierdo, "Correlation analysis of water demand and predictive variables for short-term forecasting models," *Mathematical Problems in Engineering*, vol. 2017, 12 2017.

Deep Learning for Reliable Communication Optimization on Autonomous Vehicles

J. Loureiro, J. Cecílio

Faculdade de Ciências da Universidade de Lisboa, Campo Grande 016, 1749-016 Lisboa; email: {jnloureiro, jmcecilio}@ciencias.ulisboa.pt

Abstract

Recent breakthroughs in the autonomous vehicle industry have brought this technology closer to consumers. However, the cost of self-driving solutions still constitutes an entry barrier to many potential users due to its reliance on powerful onboard computers. As an alternative, autonomous driving algorithm processing may be offloaded to remote machines, which requires a reliable connection to the cloud servers. However, despite significant 5G coverage in many countries, mobile network reliability and latency are still inadequate for this purpose. This work explores deep learning concepts to forecast signal quality as a vehicle moves, predicting when periods of degraded network quality will occur. We develop a Long Short-Term Memory (LSTM)-based neural network, trained on multivariate time series containing historical data on several mobile network parameters, and evaluate the results of multi-step Reference Signal Received Power (RSRP) prediction. Results show that our model achieves a rapidly increasing Root-Mean-Square Error (RMSE), reaching over 8 dBm after 25-time steps. This error does not allow for the accurate prediction of future signal quality.

Keywords: Deep Learning, Autonomous Vehicle, Signal Quality, Forecasting.

1 Introduction

In recent years, vehicle manufactures have been investing significantly in technology that would enable their vehicles to operate autonomously without requiring the presence of a human driver. Self-driving cars aim to improve road safety while optimizing driving performance, freeing passengers to focus on tasks other than driving. Such vehicles have the potential to revolutionize multiple transportation industries. Over the years, the technology behind autonomous vehicles has improved significantly through the evolution of control systems that process data obtained from various sensors, such as cameras, radar, lidar, sonar, and GPS. Those sensors offer the capability to collect information about the vehicle's positioning and surroundings to control and navigate the car, avoiding any obstacles. However, current autonomous vehicle technology still needs to be improved before it can be used with confidence in public roads, making it the target of large amounts of research. Sensing hardware is typically mounted on the vehicle, and these systems are usually built on machine learning algorithms and other processing-heavy mechanisms.

As technology evolves, vehicles increasingly benefit from being connected to the Internet. Such vehicles are designated connected vehicles. These connections can also be used for applications outside of autonomous driving, such as vehicle management, entertainment, and mobility management. Connected vehicle manufacturers have chosen existing mobile network technologies to power their vehicles' Internet connection capabilities. The current mobile network generation — 5G — aims to bring considerable improvements to the network's latency, which would prove extremely useful in safety-critical and real-time remote processing applications, such as offloading computer vision processing in self-driving cars.

Self-driving solutions usually require computers to be fit inside the vehicles to perform local computer vision operations. These computers use powerful CPUs and GPUs to process a large amount of information necessary to achieve autonomous driving reliably. Such computing devices are expensive, driving the vehicles' prices up. Sometimes prohibitively so, preventing a significant percentage of consumers from affording retrofitted self-driving systems or autonomous vehicles.

Alternatively, a part of the information processing can be offloaded to remote servers, reducing the computing capability requirement of onboard computing devices, thus reducing their cost. However, offloading this computing burden requires a persistent and reliable network connection between the vehicles and the servers. In fact, given the life-critical nature of cars circulating on public roads, the high reliability of the connection is vital to the safe operation of the vehicle. The network must allow all necessary data to be transmitted promptly to achieve high levels of reliability. Thus, sufficient network throughput to transmit all sensor data is indispensable to the processing offloading, as are latency values low enough to allow the vehicle to receive and react to the offloaded algorithms decisions. In addition, the car must be connected to reliable infrastructure and properly handle cell tower handovers to maintain an adequate connection quality.

Considering that the safety of the passengers is a major priority to any autonomous driving solution, maximizing it will necessarily also be of paramount importance in our work. Reducing the vehicle's speed when facing periods of network instability will improve passenger safety while allowing more

time for the data to be processed by autonomous driving algorithms. Additionally, the chosen route determines the pattern of network availability during the ride, as it influences the amount and relative position of cell towers, which would, in turn, have an impact on the vehicle's ability to offload processing power to remote devices. Choosing the route that maximizes network availability requires considering large amounts of diverse types of data, often not readily available: geographical data, cell tower position data, vehicle GPS data, and live traffic data. Moreover, route selection algorithm heuristics often value routes that minimize travel time, driven distance, and fuel consumption, which may conflict with routes with superior network performance. In this paper, we explore the concept of deep learning to forecast the communication quality while a self-driving car is moving. We leverage deep neural networks to predict the network quality metric Reference Signal Received Power (RSRP). Our forecasting model creates an opportunity to develop sophisticated control algorithms that allow improving the system's reliability and safety, and reduce its cost.

The remainder of this document is organized as follows. Section 2 describes the related work. Previous deep learning applications to forecast network parameters and methods are reviewed. Section 3 describes the dataset used, followed by an exploratory analysis of the data. We present the methods used to process and model the data in Section 4, followed by Section 5 in which we present the obtained results. Finally, Section 6 presents our preliminary conclusions.

2 Related work

Multiple research groups have previously tackled the network parameter forecasting problem using a variety of approaches. Bui et al. [1] have surveyed recent works related to network parameter forecasting. In particular, they note that there is significant interest in utilizing anticipatory mobile networking to optimize network efficiency. Additionally, numerous papers leverage historical data in the form of time series to predict network parameters, used to train a variety of machine learning models, including neural networks. Finally, they recognize the importance of research focusing on the emerging 5G technology, which will be followed by a new set of challenges, such as data collection difficulties and security considerations.

Raca et al. [2] recognize the advantages of deep learning techniques when applied to throughput prediction tasks, as it results in models requiring less storage space and providing more accurate results when compared to other machine learning algorithms. Deep learning algorithms, such as Long Short-Term Memory (LSTM)-based neural networks, are already widely used in the autonomous vehicle industry, supporting numerous self-driving solutions [3]. In fact, in the context of 5G and deep learning research, LSTM-based neural network use surpasses that of CNNs or traditional RNNs [4].

Minovski et al. [5] have used several machine learning algorithms, namely Random Forest, Support Vector Regression, XGBoost, and Multilayer Perceptron, to predict instantaneous network throughput while the mobile device is idle, based on several passive network metrics, such as signal quality and parameters from serving and neighboring cells, achieving prediction Mean Squared Errors (MSE) of 0.06 and 0.17 for LTE and 5G networks, respectively. These results suggest that network metrics are somewhat correlated and that network throughput tends to have low variance over time.

In Prihodko's work [6], signal strength (more specifically, network RSRP values) is predicted using the Vector Autoregression, Multilayer Perceptron, and Gated Recurrent Unit (GRU) algorithms to improve handover timing in mobile devices, based on past RSRP observations. The GRU-based model achieved the best results of all tested models with an RMSE of over 13.5 dBm after 10 time steps. Note that this work used simulated data to train their models and results may differ from those obtained from real-world data. In this work, they used the Rectified Linear Unit (ReLU) function as the activation function, claiming it significantly increases training speed. Conversely, we have used the hyperbolic tangent function due to internal optimizations in our chosen framework, as explained in Section 4. In addition, the ReLU function has been deemed inappropriate to use with RNNs without careful network weight initialization [7].

3 Dataset

In this work, we are using a dataset created by Raca et al. [8], which corresponds to a collection of network metric captures from mobile devices. This dataset consists of 83 traces, grouped by mobility pattern and download type, of data channel Key Performance Indicators (KPIs), captured using a network monitoring application (G-NetTrack Pro[1]) on non-rooted Android mobile devices. All traces were captured using the same mobile device, connected to a single mobile network operator. This dataset supersedes multiple previous capture collections by other research groups regarding sampling granularity, available KPIs and metrics, and scenario and technology diversity. It also replaces an older dataset published by the same authors [9] focused on 4G capture data.

Traces are grouped into categories by mobility pattern, labelled according to the primary mode of transportation. In this work, we use the 60 traces obtained while driving in urban and suburban environments around the city of Cork, Ireland, in cars. These traces correspond to approximately 95% of the total capture duration of the dataset and exhibit great diversity in KPI evolution patterns due to the high number of traces and the different congestion patterns observed during capture. The remaining traces, captured in static conditions, do not represent a relevant use case and are therefore ignored. Furthermore, traces are labeled by download type: file download, Netflix streaming, and Amazon Prime streaming. Table 1 shows the number of traces in the dataset grouped by mobility pattern and by type of download.

The following list describes the KPIs present in each sample. This list only includes the dataset features that are relevant to this work.

- *Timestamp*: sample timestamp, in the format `[YYYY].[MM].[DD]_[hh].[mm].[ss]`, with a one-second granularity.

[1] `https://gyokovsolutions.com/g-nettrack/`

Table 1: Trace distribution by mobility pattern and application pattern. Only driving traces are included.

	Download	Netflix	Amazon Prime	Total
Static	5	10	8	**23**
Car	16	23	21	**60**

- *Longitude* & *Latitude*: current GPS coordinates of the mobile device.
- *Speed*, in km/h: velocity of the mobile device, obtained from in-device GPS components.
- *NetworkMode*: sample's mobile network technology. This value depends on the highest generation standard supported by the serving cell. Valid values for this feature are *GPRS* and *EDGE* for 2G, *UMTS*, *HSUPA*, *HSDPA* and *HSPA+* for 3G, *LTE* for 4G, and *5G* for 5G.
- *RSRP*, in dBm: Reference Signal Received Power. It represents the average power observed in the Resource Elements (RE) that carry cell-specific Reference Signals (RS). REs carrying Data Signals are not considered. This metric provides a reliable indication of the connection's signal quality.
- *DL_bitrate* & *UL_bitrate*, in kbit/s: download and upload rates, respectively. During capture, the mobile devices continuously download a large file via TCP. These values represent transmission rates measured on the devices' network interface. The sum of these two values corresponds to the device's network throughput.

Traces representing the download of large files show higher throughput values when compared to the other two application patterns, corresponding to video streaming from popular online streaming services. The applications used to download large files attempt to achieve the shortest download time using the network's highest throughput capacity. Conversely, video streaming applications download video files in segments, remaining idle while the video buffers are full. In addition, these applications' HTTP Adaptive Streaming (HAS) techniques keep the application throughput below the network's maximum capacity to ensure a satisfactory user experience. This results in throughput values for streaming applications that are consistently lower than for the file downloads. In our use case, the throughput profile would resemble a streaming download since the amount of data in the transmission is limited by the data obtained from the sensors over time. However, the data size would often push the network throughput capacity to its limit. We then argue that samples from both file download and streaming profiles should be used in the forecasting process. Ping statistics are missing in over 98% of observations. This prevents latency values (extracted from average ping values) from being used in model training. Calculating the Pearson correlation coefficient for each significant numerical feature pair, we found no strongly correlated features.

4 Forecasting Network design

Each observation in the dataset is labelled with a sequential timestamp with a consistent time step of one second. Therefore, all samples are time series, making RNNs one of the most appropriate neural network classes to process the data involved in this work. For this reason, LSTMs will be used for forecasting network quality.

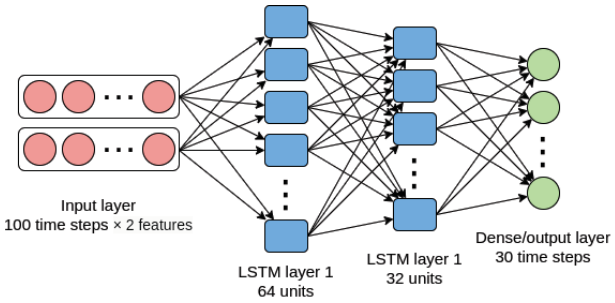

Figure 1: Neural network architecture diagram.

The proposed solution is divided into three layers. The first layer is an LSTM layer with 64 units that consume the input data. This layer is configured to return a data matrix instead of a vector, which a second LSTM layer will process with 32 units. Both layers use the hyperbolic tangent function as an activation function and the sigmoid function as a recurrent activation function, which allows model operations to be executed using the cuDNN[2] GPU-accelerated library, significantly accelerating training and prediction operations. Then, there is a dense layer of 30 neurons corresponding to the output neurons. It is used to change the dimension of the output vector. The model is configured with the Adaptive Moment Estimation (Adam) optimizer and the Mean Squared Error (MSE) loss function, causing higher error values to impact the weight adjustment process significantly. Figure 1 illustrates the model's layer architecture. All numerical feature data is converted to floating-point representations and normalized using a minimum-maximum scaler to the range 0 to 1 before being input into the neural network.

The two features used to train the model are throughput, calculated from the download and upload bitrates, and RSRP. The target feature to forecast is RSRP, corresponding to the device's signal strength. The model was trained using 100 past time steps, with one additional feature besides the target feature, and 30 future time steps for each sample. The samples were then split into training and test data at a ratio of 9:1. This resulted in 86170 training samples and 9575 test samples. Forecasting 30 time steps in the future with a granularity of one second corresponds to a prediction range of 30 seconds, or 500 meters at 60 km/h and 1000 meters at 120 km/h. These are common speeds for vehicles traveling in urban and highway environments, respectively, and the prediction ranges would allow enough time to react to periods of degraded signal quality. Model training was run for 10 epochs using batches of 32 samples.

5 Results

The model is evaluated using Root-Mean-Square Error (RMSE), a commonly used evaluation metric that penalizes more significant deviations from the mean. In the context of this work, higher error values are particularly pernicious

[2] https://developer.nvidia.com/cudnn

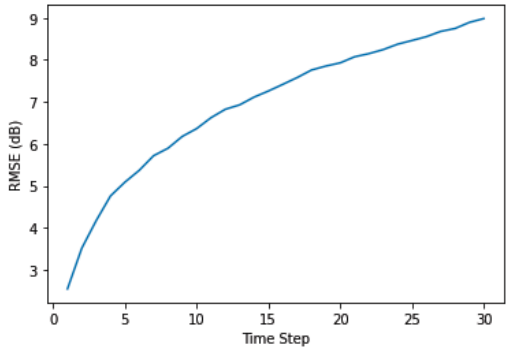

Figure 2: Prediction RMSE for each predicted time step.

due to the criticality of the application. Sizeable forecasting errors lead to catastrophic vehicle control decisions, which may ultimately result in loss of passenger life. The trained model was used to forecast RSRP values for both training and test data. Our experiments achieved an RMSE of 7.1051 dBm for the training data and 7.1297 dBm for the test data. Similar bias and variance suggest that the model is not underfitting the test data, showing nearly indistinguishable results for previously seen and unseen data. The error for training data ranges from 2.6749 dBm to 8.9956 dBm for the first and last (30th) predicted time steps, respectively. We got errors from 2.5448 dBm to 8.9851 dBm concerning the testing data. As evidenced in Figure 2, the error proliferates as the prediction gains distance from the moment of the last observation. This growth resembles a logarithmic curve that decelerates as more time steps are forecast. In practice, given the obtained error amplitudes, the predictions resulting from the current model could not be used with sufficient confidence to enable reliability for a control algorithm to operate in an autonomous vehicle. We considered that the values are acceptable for one or two time-steps from our preliminary results. Still, when we predict a considerable period, we get significant errors decreasing the model's usefulness.

All data processing and model training was run using the Google Colab[3] platform's cloud computing resources, on machines with the following specifications: Intel Xeon processor running at 2200MHz, 12GB of RAM, Nvidia Tesla K80 GPU with 12GB of VRAM, running Ubuntu Linux version 18.04.5 LTS. All implementation and visualization code was written in Python, and the neural network was implemented using the Keras[4] framework.

6 Conclusion

In this work, we explored the concept of deep learning to forecast the communication quality of a self-driving car when it is moving. We designed a deep neural network to predict the RSRP received by a vehicle several steps onwards. Our preliminary results concluded that the mobile network metric RSRP could not be accurately forecast based on patterns extracted from previous observations of network KPIs such as RSRP and throughput. Intuitively, previous traces have a low correlation with future network usage experience in previously unseen locations, as characteristics of the area surrounding the mobile device in the past cannot infer characteristics of future surrounding areas. Successful forecasting of network parameters requires more information related to the vehicle's surroundings along its planned route, such as morphology data, including nearby building boundaries and cell towers within reach of the vehicle's antenna. When successfully applied, these forecasting models create an opportunity to develop sophisticated control algorithms that improve the system's reliability and safety and reduce its cost.

[3] https://colab.research.google.com/
[4] https://keras.io/

Acknowledgments

This work was supported by the LASIGE Research Unit (ref. UIDB/00408/2020 and ref. UIDP/00408/2020), and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 957197 (VEDLIoT project).

References

[1] N. Bui, M. Cesana, S. A. Hosseini, Q. Liao, I. Malanchini, and J. Widmer, "A survey of anticipatory mobile networking: Context-based classification, prediction methodologies, and optimization techniques," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1790–1821, 2017.

[2] D. Raca, A. H. Zahran, C. J. Sreenan, R. K. Sinha, E. Halepovic, R. Jana, and V. Gopalakrishnan, "On leveraging machine and deep learning for throughput prediction in cellular networks: Design, performance, and challenges," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 11–17, 2020.

[3] J. Ni, Y. Chen, Y. Chen, J. Zhu, D. Ali, and W. Cao, "A survey on theories and applications for self-driving cars based on deep learning methods," *Applied Sciences*, vol. 10, no. 8, p. 2749, 2020.

[4] G. L. Santos, P. T. Endo, D. Sadok, and J. Kelner, "When 5g meets deep learning: a systematic review," *Algorithms*, vol. 13, no. 9, p. 208, 2020.

[5] D. Minovski, N. Ogren, C. Ahlund, and K. Mitra, "Throughput prediction using machine learning in lte and 5g networks," *IEEE Transactions on Mobile Computing*, 2021.

[6] N. Prihodko, "Machine learning for forecasting signal strength in mobile networks," 2018.

[7] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.

[8] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: A 4g lte dataset with channel and context metrics," in *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, (New York, NY, USA), p. 460–465, Association for Computing Machinery, 2018.

[9] D. Raca, D. Leahy, C. J. Sreenan, and J. J. Quinlan, "Beyond throughput, the next generation: A 5g dataset with channel and context metrics," in *Proceedings of the 11th ACM Multimedia Systems Conference*, MMSys '20, (New York, NY, USA), p. 303–308, Association for Computing Machinery, 2020.

Compiler Support for an AI-oriented SIMD Extension of a Space Processor

Marc Solé, Leonidas Kosmidis

Barcelona Supercomputing Center (BSC) and Universitat Politècnica de Catalunya (UPC); email: {marc.solebonet, leonidas.kosmidis}@bsc.es

Abstract

In this on going research paper, we present our work on the compiler support for an AI-oriented SIMD Extension, called SPARROW. The SPARROW hardware design has been developed during a recently defended, award-winning Master Thesis and is targeting Cobham Gaisler's space processors Leon3 and NOEL-V. We present the compiler support we have included in two compiler toolchains, gcc and llvm as well as a SIMD intrinsics library for easy programmability. Compiler modifications are kept to minimum in order to enable incremental qualification of the toolchains. We present our experience working with the two compilers and performance results for the two compilers on top an FPGA implementation of the target space processor.

Keywords: compiler, SIMD, AI, space processor.

1 Introduction

In recent years, artificial intelligence (AI) and related topics, such as machine learning (ML) and neural networks (NN), have been explored in many different fields. Space systems are not an exception; the advantages that AI applications can provide in space operations are numerous, thus there are many on-going efforts to accelerate AI processing in space.

The simple, in-order, low-power processors traditionally used in space systems cannot meet the increased performance demands of AI. In such a critical environment, real-time capabilities and space qualification are mandatory properties, which are costly to provide in completely new designs.

While commercial off-the-shelf (COTS) AI accelerators and embedded GPUs have been used as an alternative in certain cases such as experimental missions and nano-satellites, they are not a definitive solution for high-risk missions. COTS accelerators are not radiation tolerant – a requirement to work beyond low-earth orbit – nor they have appropriate software stacks for the applications in space or support for real-time operating systems.

For this reason, in this recently presented Master's thesis project, we implemented SPARROW [1], a small, open-source SIMD module to accelerate the computation of AI applications in an already qualified, widely used space processor, LEON3, with minimal hardware and software changes. It is directly connected into the integer pipeline and provides additional vector instructions to improve such applications.

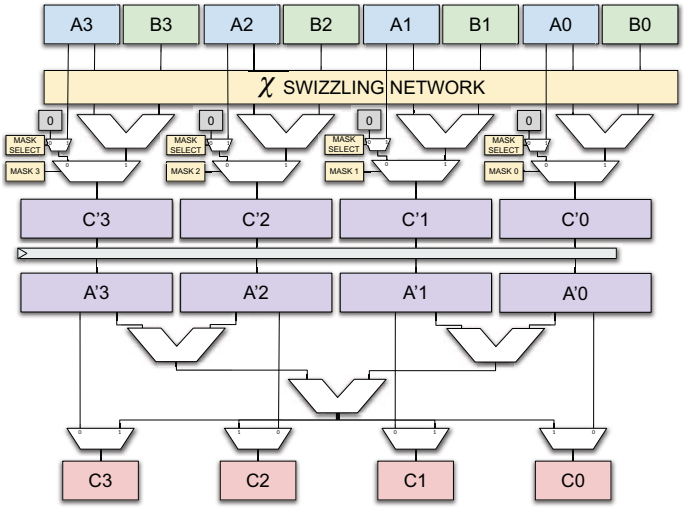

Figure 1: Overview of the SIMD SPARROW module [1]

The hardware cost of the module is minimal compared with conventional vector approaches, thanks to the re-utilization of the integer register file. This is possible since 8-bit operations have been shown enough for AI applications in the literature and in commercial AI hardware. Therefore, each integer register can work as a vector with up to 4 8-bit components. To our knowledge, this is a unique feature of our work. Further advantages of our choice is the simplification of data management, which eliminates the need for new load-store instructions, allowing a small incremental qualification cost of the hardware and its compiler. More details on the hardware design of the module are provided in [1].

In this work in progress paper, we describe our on-going work regarding the addition of software support for SPARROW with two widely used compilers, gcc and llvm. We describe our experience working with these two compilers and the development of small preprocessor library which allows to program SPARROW in a similar way with SIMD intrinsics for other processors. In addition, we provide some early comparison results of the performance of the two compilers both with handwritten assembly implementations as well as with our SIMD library.

2 Background on SPARROW Design

Before we discuss about the compiler support, we first need to briefly describe the SPARROW hardware design. The SPARROW SIMD unit is co-designed analyzing the most important features and characteristics of ML workloads. The

```
1  unsigned char weights[32*32];
2  unsigned char next_layer[32*32];
3  unsigned int a, b, result, scr;
4
5  /* set the value of the %scr */
6  scr = 0x0D9D0E; //swizzling_B = 1-2-3-0,
7                  // swizzling_A = 3-2-2-0,
8                  // mask_select = 0, mask = 1110
9
10 asm("wr %0", %%scr : : "r"(scr));
11 /* initialise all a components to 0, ie a.xyzw=0 */
12 a = 0;
13 /* b.xyzw = weights[0].xyzw */
14 b = *((unsigned int*) &weights[0]);
15 asm("nop"); // wait for %scr to commit the write
16 /* result.xyz = a.xyy + b.zyx */
17 asm("usadd_ %1, %2, %0": "=r"(result) : "r"(a), "r"(b));
18 /* next_layer[0].xyzw = result.xyzw */
19 *((unsigned int*) &next_layer[0]) = result;
```

Figure 2: Example of SPARROW programming in C with inline assembly

```
1  unsigned char weights[32*32];
2  unsigned char next_layer[32*32];
3  unsigned int a, b, result, scr;
4
5  /* set the value of the %scr */
6  __sparrow_setMask(0b1110);
7  __sparrow_setMaskSel(0);
8  __sparrow_setSwizzlingA (3,2,2,0) ;
9  __sparrow_setSwizzlingB (1,2,3,0) ;
10
11 __sparrow_writeSCR();
12 /* initialise all a components to 0, ie a.xyzw=0 */
13 a = 0;
14 /* b.xyzw = weights[0].xyzw */
15 b = *((unsigned int*) &weights[0]);
16 asm("nop"); // wait for %scr to commit the write
17 /* result.xyz = a.xyy + b.zyx */
18 __nop(result, a, "usadd", b);
19 /* next_layer[0].xyzw = result.xyzw */
20 *((unsigned int*) &next_layer[0]) = result;
```

Figure 3: Example of SPARROW programming in C for SPARC v8 with the SPARROW SIMD library

dot product is one of the most frequently used computations in ML as it is used in matrix multiplication which is a recurrent kernel in the computation of NN for fully connected layers and convolutions. SPARROW features a 2-stage approach which allows to compute a dot product with a single instruction as can be seen in Figure 1. In the first stage, for vector-vector operations, the data are computed in parallel allowing up to 4 simultaneous operations. In the second stage, reduction operations are performed on the result from the first stage. Both stages can be bypassed in order to just perform any of the two types of operations. This allows up to 200 combinations of operations such as addition, multiplication, maximum and minimum, bitwise operations, etc with just the introduction of 13 vector instructions in the first stage and 4 reduction operations in the second one. Both signed and unsigned version are included.

Additionally, SPARROW includes GPU-like features to provide even more flexibility to the module, such as masking and swizzling. Both characteristics can be controlled by using a special register, the SPARROW control register (%scr). As other special registers it can be accessed using the already existing instructions in the ISA. To avoid overflow, SPARROW also includes a saturation version of the instructions, both signed and unsigned, which clips the results at 0 to 255 for unsigned or at -128 to 127 for signed.

The module is highly portable and can be used in different base processors with minimal modifications. Currently SPARROW has been integrated with the SPARCv8-compliant LEON3 processor, developed by Cobham Gaisler, preserving its 100MHz frequency. The module has also been ported to the NOEL-V processor, a RISC-V based space processor by Cobham Gaisler, as we discussed in our talk at the RISC-V Forum: Vector and Machine Learning [2] [3]. SPARROW won the fist position in Xilinx's Open Hardware Competition 2021 in the student category, and awarded the best Master Thesis in Spain for 2021 by the Spanish IEEE AESS Chapter.

3 SPARROW Software Support

3.1 Compiler support

An important advantage of SPARROW compared to custom accelerators is the ability to reuse the existing qualified software stack of LEON3 i.e. the RTEMS real-time operating system or bare-metal space applications, which reduces both the cost and the effort of the development of a new compiler from scratch as well as its qualification cost later.

We added SPARROW support in the two most widely used compilers nowadays, gcc and llvm. We modified the `binutils` of Gaisler's `bcc-2.2.0` gcc-derivative compiler and the base `LLVM v13.0`. We use an underscore as separation between the instruction names of the two stages. In the case of `nop` it is omitted for the second stage. An `s` and `u` prefix in the instruction name denotes saturation and unsigned operation, i.e. `usmul_`. We also added aliases such as the dot product, which can be both represented by `mul_sum` or `dot`. The SPARROW control register can be accessed using the `wr`, `rd` and `mov` instructions present on the SPARC v8 ISA for accessing special registers.

We program SPARROW in C, using inline assembly instructions as shown in the example of Figure 2 for a saturated vector addition with unsigned 8-bit values using swizzling and masking. As it can be seen, SPARROW can be programmed in a high level way, not very different than vector intrinsics for conventional SIMD extensions such as NEON.

Another important advantage of reusing the integer register file is that we can use the regular load and store instructions. Inline assembly (and code generation) is only required for the SIMD operations. Moreover, this means that we don't need to specify explicit registers in the inline assembly, nor to modify the compiler register allocator. Notice that by passing the integer variable names to the inline assembly instruction (line 17), the SIMD instruction accesses directly the register in which each of the variable is allocated by the compiler.

3.2 SPARROW SIMD Library

A disadvantage of programming SPARROW in assembly is that there are features like the mask and swizzling which the programmer needs to be aware of. In order to make the setting of the SPARROW Control Register transparent, we decided to create a library that contains multiple definitions to simplify working with SPARROW in SIMD intrinsics fashion.

Function	Description
`__sparrow_readSCR(X)`	Stores the current value of the SPARROW Control Register in the variable `X`
`__sparrow_writeSCR()`	Writes in the SPARROW Control Register the value of `__sparrow_scr`
`__sparrow_set(X,Y)`	Writes in the SPARROW Control Register `X xor Y`
`__sparrow_resetSCR()`	Resets the value of the SPARROW Control Register
`__sparrow_setMask(X)`	Sets the mask bits of `__sparrow_scr` to `X`
`__sparrow_setMaskSel(X)`	Sets the mask selection bit of `__sparrow_scr` to `X`
`__sparrow_setSwizzlingA(X,Y,Z,W)`	Sets the first operand swizzling order in `__sparrow_scr` to `X-Y-Z-W`
`__sparrow_setSwizzlingB(X,Y,Z,W)`	Sets the second operand swizzling order in `__sparrow_scr` to `X-Y-Z-W`
`__sparrow_(op1, op2, A, B, C)`	Performs the `op2` reduction on `A op1 B` and stores the value in `C`
`__nop(C, A, op1, B)`	Computes `C = A op1 B`
`__sum(C, A, op1, B)`	Computes a sum over `A op1 B` and stores the result in `C`
`__max(C, A, op1, B)`	Computes the maximum in `A op1 B` and stores the result in `C`
`__min(C, A, op1, B)`	Computes the minimum in `A op1 B` and stores the result in `C`
`__xor(C, A, op1, B)`	Computes a xor reduction over `A op1 B` and stores the result in `C`
`__usum(C, A, op1, B)`	Computes an unsigned sum over `A op1 B` and stores the result in `C`
`__umax(C, A, op1, B)`	Computes the unsigned maximum in `A op1 B` and stores the result in `C`
`__umin(C, A, op1, B)`	Computes the unsigned minimum in `A op1 B` and stores the result in `C`

Table 1: SPARROW library functions

The SPARROW SIMD library is implemented using C-preprocessor macros that convert function-like calls into the inline assembly. For the SCR, a variable is declared which is modified when setting the mask and swizzling and is used to write in the special register. One of the advantages of having a library implemented like this is once again portability and simplicity. Table 1 shows the existing functions in the SPARROW library.

In Figure 3 the same code shown in Figure 2 is represented using the SPARROW library. Note that the setting of the SCR, which starts at line 6, requires more instructions, however since those are C-preprocessor macros the compiler can reduce the number of actual generated instructions. On the other hand, although the same behaviour as with the inline assembly could be achieved by using `__sparrow_setSCR(X,Y)`, with this approach the value of each field is more clear and the programmer does not require any knowledge on the SCR organization. In line 11 it is necessary to include the writing of the SCR as the previous lines were just setting the library internal variable. This is done to reduce the number of accesses which otherwise, would be necessary if each line performed the actual write.

4 Preliminary Experimental Results

We have evaluated our compilers and SIMD library on an FPGA implementation of SPARROW integrated with LEON3 [4]. For our preliminary evaluation we are using a widely used and well understood kernel, matrix multiplication, which is an essential block for AI inference applications, since it is used for representing fully connected layers as well as for the implementation of convolutions. In addition to the sequential version of matrix multiplication written in C which is used as a baseline, we have produced two additional versions of the code, one written in SPARROW assembly and one using the SPARROW SIMD library. Moreover, we have developed two variants of matrix multiplication, one using saturation and one allowing the values to wrap around when an overflow occurs. However, due to space reasons we only provide results with the version with saturation, since the results for the other version exhibit the same trends, but with lower speedup ranges (from 2.1× - 6.8×). All programs are compiled using the highest optimisation level (-O3).

Figure 4 shows the comparison between the various versions, for different size of matrices, for common matrix sizes found in machine learning applications. The results are normalised with respect to the gcc sequential (CPU) version, which is also shown in the figure with a red line at value 1. As a consequence, higher values are better and values over the red line represent speedup, while values below it show slowdown.

First we compare the sequential versions of the two compilers. For the two smaller sizes (4 and 8), gcc provides slightly faster code than llvm, while for sizes 16 and 32 llvm is faster. However, when the size of matrices exceed the size of the data cache (8KB), the performance difference is negligible.

When comparing the SIMD assembly versions, we notice that gcc generates faster code than llvm for all sizes, achieving a maximum speedup of 17.3× over the sequential version for matrix size 32. The llvm still provides a good speedup compared to the sequential version, up to 15.2×, being only 10% slower that gcc.

The SIMD library inevitably incurs some overhead compared to the assembly implementations, especially in the case of gcc. However, llvm provides the same performance with the version that uses assembly instead of the library for sizes of 128 and larger. Finally, comparing the SIMD library versions on top of gcc and llvm, gcc provides the same performance with llvm, except in sizes 8 and 16, where it is slightly faster.

5 Lessons Learnt

Having worked with both GCC and LLVM for the development of the software support for SPARROW has allowed us to compare, not only the performance, but also the experience when working with each one. It is worth noting that we had no prior experience on working on either of the two toolchains

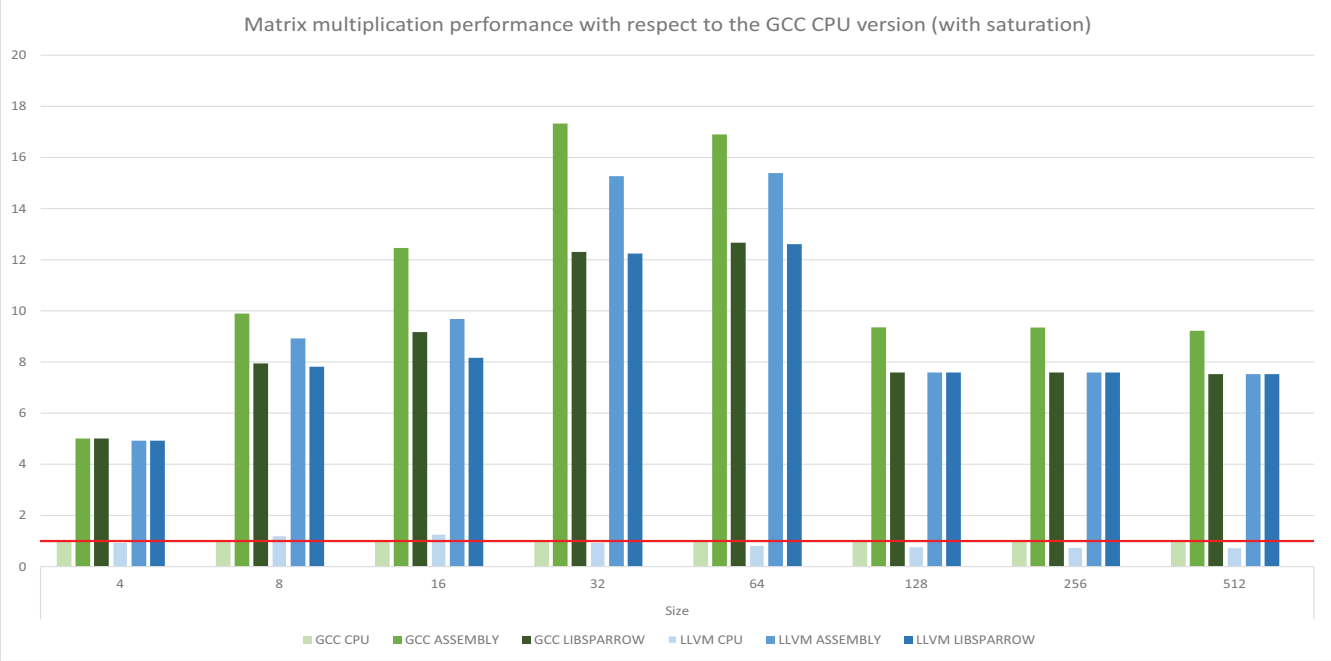

Figure 4: Plot of the performance for matrix multiplication with saturation enabled

prior to this project. In the previous section we already presented a preliminary performance comparison between the two compilers. A detailed analysis would require more experimentation and the evaluation in different scenarios. In general, however, it is shown that gcc-compiled executables have lower execution times, at least for LEON3.

When working to include the SPARROW assembly instructions one of the key advantages of llvm over gcc was the possibility of defining the instructions in a nested way. This simplifies the addition of two-stage instructions allowing a simpler combination of them. However, a new line for each combination must be manually added. On the other hand, the code for adding these instructions is easier to understand in GCC, which can be easily deduced from the existing instructions. Fortunately, LLVM has a great documentation and a large number of tutorials about on how to modify it.

All in all, both compilers had advantages and disadvantages compared to one another, however they both offer good facilities to implement the required functionality in order to add software support in new hardware designs.

6 Conclusions and Future Work

In this paper we have presented our on-going work on the addition of software support for the SPARROW SIMD unit in the gcc and llvm compilers, and a SIMD library that facilitates SPARROW programming without using inline assembly. In terms of development both compilers had advantages and disadvantages compared to one another, however they both offer good facilities to implement the required functionality in order to add software support in new hardware designs.

In terms of performance, we noticed that gcc provides higher performance when sequential C code or assembly is used, but both compilers provide similar performance when our SIMD library is used. As a future work we want to perform an extensive evaluation of the compiler backends we developed as well as of our SIMD library, by porting more applications in SPARROW. Moreover, we would like to evaluate Ada's frontends of both compilers, generate an Ada version of our SIMD library and compare them among them and with the C frontends. Finally, we plan to add support in the compilers so that they can generate directly SPARROW assembly instructions, through autovectorisation.

7 Acknowledgments

This work was funded by the Ministerio de Ciencia e Innovacion - Agencia Estatal de Investigacion (PID2019-107255GB-C21/AEI/10.13039/501100011033 and IJC-2020-045931-I) and partially supported by the European Space Agency (ESA) through the GPU4S (GPU for Space) activity and the HiPEAC Network of Excellence.

References

[1] M. Solé and L. Kosmidis, "SPARROW: A Low-Cost Hardware/Software Co-designed SIMD Microarchitecture for AI Operations in Space Processors," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2022.

[2] Linux Foundation, "RISC-V Forum: Vector and Machine Learning." https://events.linuxfoundation.org/riscv-forum-vector-and-machine-learning.

[3] Marc Solé, Leonidas Kosmidis, "RISC-V Forum: Vector and Machine Learning." https://events.linuxfoundation.org/riscv-forum-vector-and-machine-learning/program/schedule/.

[4] M. Solé and L. Kosmidis, "SPARROW source code repository," 2021. https://gitlab.bsc.es/msolebon/sparrow.

Space Compression Algorithms Acceleration on Embedded Multi-core and GPU Platforms

Alvaro Jover-Alvarez*, Ivan Rodriguez*, Leonidas Kosmidis

Barcelona Supercomputing Center (BSC) and Universitat Politecnica de Catalunya (UPC); email: {ajover, irodrig, lkosmidi}@bsc.es

David Steenari

European Space Agency (ESA); email: David.Steenari@esa.int

Abstract

Future space missions will require increased on-board computing power to process and compress massive amounts of data. Consequently, embedded multi-core and GPU platforms are considered, which have been shown beneficial for data processing. However, the acceleration of data compression - an inherently sequential task - has not been explored. In this on-going research paper, we parallelize two space compression standards on both CPUs and GPUs using two candidate embedded GPU platforms for space showing that despite the challenging nature of CCSDS algorithms, their parallelization is possible and can provide significant performance benefits.

Keywords: embedded GPUs, multi-core, space compression.

**Both first authors contributed equally to the paper.*

1 Introduction

The on-board processing requirements of future space missions are constantly increasing, requiring new hardware to satisfy this need.Embedded COTS platforms featuring multi-core CPUs and GPUs are promising candidates, combining high-performance and low power consumption. The GPU4S (GPU for Space) ESA-funded project [1] studies whether on-board processing algorithms are amenable to GPU parallelization as well as whether embedded GPUs can satisfy the performance requirements of future space missions, effectively paving the way for their adoption.

However space compression algorithms are among the most challenging space processing algorithms in order to be parallelized, due to their inherent sequential nature, created by data dependencies.

Due to the importance of data compression, current spacecraft include specific ASIC or FPGA implementations of the various space compression CCSDS standards for supporting these tasks. However, an efficient parallel software implementation of these standards targeting embedded multi-core CPUs and GPUs can allow a series of benefits for future space missions.

There are 3 main families of space compression standards defined by the Consultative Committee for Space Data Systems (CCSDS) which consists of representatives from several space agencies and private corporations: CCSDS 121 covers general purpose data compression in a lossless way, CCSDS 122 covers both lossless and lossy image compression and CCSDS 123 focuses on hyper-spectral lossless and near-lossless compression. Early project results in GPU4S with commonly used processing algorithms [2] indicate that embedded GPUs can provide significant processing improvements of several orders of magnitude compared to existing space processors such as LEON/SPARC. Compared to FPGAs, which are commonly used in on-board processing applications, GPUs offer the capability to reconfigure the on-board processing using software in a fast manner.

We present our work on the acceleration of two of the most widely used space compression standards nowadays, CCSDS 121.0-B-3 [3] and CCSDS 122.0-B-2 [4] using parallel embedded COTS platforms which are considered good candidates for on board processing in the future.

Our results on two embedded platforms with multicore CPUs and GPUs, the NVIDIA Xavier and the AMD Embedded Ryzen V1605B show that the parallelization of space compression algorithms for on-board processing is possible and comparable with existing space solutions. Our implementations are available as open source, as part of ESA's OBPMark (On-Board Processing Benchmark) benchmarking suite [5], focusing on the evaluation of general purpose devices for upcoming space missions, using complex applications relevant to the space domain.

2 Parallelisation approaches

2.1 CCSDS 121.0-B-3

The CCSDS 121.0-B-3 [3] implements a lossless compression of 1D data. The algorithm architecture consists of two blocks, a *preprocessor* and an *adaptive entropy encoder*.

The preprocessor step is optional and can be omitted. It applies a reversible function to the input data to remove the correlation between its values and to convert them in a probability distribution. The predictors work with a block size J parameter which needs to be provided for the decompression.

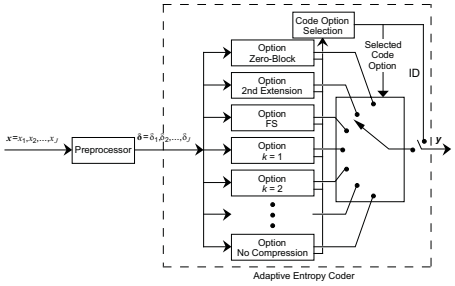

Figure 1: Structure of the CCSDS 121.0-B-3 [3] space compression algorithm. Image courtesy of [3].

The adaptive entropy encoder selects a different encoding of the preprocessed input based on the input data distribution. In fact, the input data are encoded using all the encoders implemented in the adaptive encoder, and the best one is selected i.e. the one which results to a higher compression ratio for each input data block.

The standard defines a series of encoders which can be used: Zero-Block, Second-Extension, Fundamental Sequence, Sample Splitting and No-compression (Figure 1). The Sample Splitting encoder is parametric based on a value k, with $1 \leq k \leq 29$, resulting to a total of 33 possible encoders. Note that similar to the preprocessor, not all encoders are required to be provided by a compliant implementation.

For the parallelization we follow a coarse-grain approach for both the CPU and GPU. In the CPU implementation, which is based on OpenMP, we distribute input data blocks of size J to each of the CPUs in the system. Each of the CPUs applies the entire pipeline shown in Figure 1 on its provided data, and outputs its selected compression encoder and its compressed data.

Our GPU parallelization uses a similar approach implemented in CUDA for the NVIDIA and in OpenCL for the AMD platform. However, instead of distributing the image blocks to the CPUs, we are distributing them in separate *Streams*/*Command Queues*. This version requires more frequent synchronisations compared to the CPU version, in order to synchronise between different kernel invocations, which are more costly.

2.2 CCSDS 122.0-B-2

The CCSDS 122 [4] standard provides lossless or lossy 2D data compression based on the Discrete Wavelet Transform (DWT). It consists of two main functional blocks, the Discrete Wavelet Transform and a Bit-Plane encoder.

The purpose of the Discrete Wavelet Transform is to decompose the input to a high and low frequency components to decorrelate the input data before the encoding.

The standard uses 3 levels of 2D DWTs, each of which we compute in parallel on both the multicore and GPU implementations by applying the one dimensional DWT first in rows and then in columns. For each level, the process is repeated for the top left part of the image in a pyramidal fashion. The remaining values on the top left corner are the ones containing the highest quantity of information and are called DC coefficients. The rest of the values which only add extra information are called AC coefficients. Both lossless and lossy compression can be achieved with this method, by using an integer approximation or floating point version of the transform with higher compression achieved by the latter. We obtained similar performance for both lossless and lossy compression for each parallel implementation.

The bit planar encoder encodes the coefficients of the decomposed image in blocks consisting of coefficients which correspond roughly to a region of the input image. When the integer transform is used, the encoder exploits information about least significant bits of certain frequency components being 0 as a result of their scaling. DC and AC components follow a different encoding scheme but despite their different encoding algorithms, in both cases their characteristics are taken into account in order to increase the compression ratio, such as the dynamic range they represent. The selected encoding method is specified in the output to enable its reconstruction later. Like CCSDS 121, it is possible that values remain uncoded, especially if this minimizes the number of required bits. If different component encodings require the same number of bits with the uncoded option, the standard mandates the use of the uncoded one.

3 Experimental Results

3.1 Experimental Setup

We execute our implementations on two embedded SoCs featuring multiple CPUs and GPUs, the NVIDIA Xavier and the AMD Embedded Ryzen V1605B. These two embedded platforms are the latest embedded GPUs of these vendors and have been identified as good candidates in terms of theoretical performance and power consumption, by multiple independent studies of using GPUs in space [6] [7] [8] [9] [10] and they are considered for further evaluation of their properties.

Both boards have similar characteristics, and we are using them with 4 enabled CPUs since in the case of the NVIDIA Xavier the manufacturer ensures that the board maximum power consumption is capped at 15W, which has been identified as a limit for on-board processing hardware [7]. For the AMD board such information is not provided by the manufacturer, but it is configured with the same properties for fair comparison. Both boards use Ubuntu 18.04 LTS. However, it is worth to note that OpenCL is not currently supported by AMD out of the box, so we are using a custom driver provided by Bruhnpace AB [11], which might not be as optimized as a driver provided by the GPU vendor.

3.2 Results

For the performance evaluation of our algorithms we report both execution times as well as MSamples/s and MPixels/s which are the standard metrics used in the literature. During the execution we measure the voltage and the current and report the maximum power consumption of the boards for each experiment.

As we have mentioned, our GPU implementations are parametric, so the number of streams and thread block sizes are configurable. We tune these parameters in order to select the values that provide the best performance. For the CPU version, OpenMP automatically uses the number of available cores, which is 4 in both boards.

3.2.1 *CCSDS 121*

For the evaluation of our CCSDS 121 implementations, we use the standard methodology followed by other works in the literature, using randomly generated data. In particular, we use 16 MB of randomly generated data which is divided in 1024 Steps, each of which consists of 256 blocks of 64 bytes. Moreover, we ensure that all the compared implementations use the same input data and produce identical output.

Figure 2 shows the results between the sequential implementation and our parallel versions for both platforms for various Block Sizes (J values). We notice that the sequential CPU version is faster in the AMD, which means that the Embedded Ryzen x86 CPU has higher performance than the NVIDIA designed "Carmel" ARM v8.2 CPU. Similarly, the OpenMP version is faster in AMD than in the NVIDIA platform. In both cases, we see a speedup of the parallel version compared to the sequential one, 81% in the Xavier and an impressive 2.2× in the Emdedded Ryzen.

Regarding the GPU performance, the NVIDIA GPU provides a speedup of up to 2.1× over the sequential version but equivalent or lower performance than the parallel CPU version on the same platform. However, in the AMD platform, the GPU version provides similar performance with the sequential version and it is 2× slower than the parallel CPU version on the same platform.

We have identified that the reason of low GPU performance comes from the use of atomic operations. We are using the atomics operations for the implementation of the ZeroBlock encoder. Due to the high overhead of atomics on the AMD platform, ZeroBlock becomes the bottleneck of the GPU implementation. In NVIDIA GPUs, each hardware generation reduces the cost of atomics operations. However, in AMD GPUs such information is not available.

The AMD platform contains a more powerful CPU than the Xavier, as it can be seen in terms of absolute performance (Mpixels/s) for the sequential version. This in addition to the fact that the overhead of atomics is smaller in the CPU, due to smaller number of threads translates to exceptional multicore performance, which is faster than the GPU implementations of both platforms.

Moreover, we observe that the maximum performance is obtained for block size 16. Our multi-core performance on the AMD V1605B is close to the requirement of 60 MSamples/s which is usually a target for space applications, as specified by a recent ESA funded FPGA development project which resulted to the best state-of-the-art CCSDS 121 hardware implementation [12]. Table 1 shows the power consumption for the same experiments. However, there are no reported power consumption data for state-of-the-art CCSDS 121 implementations for comparison.

3.2.2 *CCSDS 122*

For the evaluation of our parallel implementations in this compression algorithm, again we have followed the standard practice used in the literature for the performance evaluation of this algorithm implementations, using uniform images such as completely black, random generated images and real

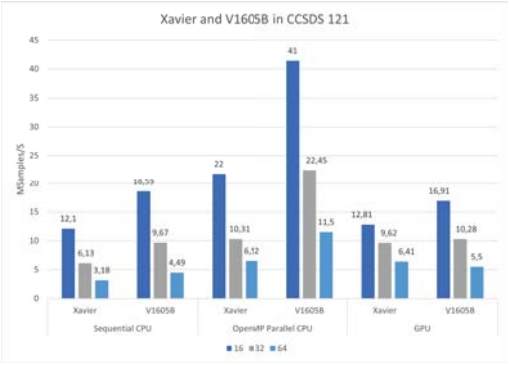

Figure 2: CSDS121 performance in MSamples/s for the NVIDIA Xavier and V1605V fixed at 1024 steps with 256 sample intervals for different Block Sizes J.

J Size	CUDA	Sequential	OpenMP
16	9.16 W	9.35 W	9.67 W
32	9.28 W	9.03 W	9.83 W
64	9.03 W	9.40 W	9.89 W

Table 1: Measured power consumption when running CCSDS 121 with different J size on the NVIDIA Xavier platform.

images from space missions. In particular, we have employed several synthetic and real images of different sizes, such as one from NASA's Mars Pathfinder Mission from the area surrounding Yogi [13] (a rock named by Geoffrey A. Landis) and one from NOAA, taken by the Metop C satellite on 2019-12-21 during its ascending orbit direction.

Table 2 shows the comparison of our implementations using the floating point implementation of the DWT. Although we implemented both integer and floating point version of the DWT, the results are very similar, so we only show one of them. The same trends observed with the CCSDS 121 implementations are visible in this version, too, however on a different scale. In this algorithm, the OpenMP implementations are only slightly faster than the sequential versions, and the AMD CPU is faster than the ARM CPU. On the other hand, the NVIDIA GPU provides a 10× performance benefit compared to the sequential version. However, the AMD GPU is significantly slower (4×) than the sequential CPU version.

According to our analysis, the bottleneck in the AMD GPU implementation comes from additional memory copies and the conversion operations between floating point and integers, which are very costly in the AMD platform. These operations are performed just before and after the DWT stage. As an indication, in the integer version of DWT for the 122, the copy operation takes 3s, while in the Xavier 20ms. Similarly, in the floating point version, in the AMD platform the memory copy and conversion takes 9.4s, while in the Xavier it takes 300ms. Again, we don't know if this is a hardware issue e.g. if the memory bandwidth of the AMD platform is lower than the one of the Xavier, or an issue of the unofficial driver e.g. the driver does not use DMA for the memory transfers, or does not overlap kernels and memory copies. Of course, the sequential version of the algorithm does not require these very

Image	Secuential (Xavier)	Sequential (V1605B)	OpenMP (Xavier)	OpenMP (V1605B)	CUDA (Xavier)	OpenCL (V1605B)
NOAA Image	5.48 Mp/s 9.07 W	7.194 Mp/s ∼15 W	5.96 Mp/s 9.23 W	7.918 Mp/s ∼15 W	6.466 Mp/s 11.48 W	0.906 Mp/s ∼15 W
Mars marspath	6.16 Mp/s 9.02 W	7.434 Mp/s ∼15 W	5.01 Mp/s 8.96 W	7.204 Mp/s ∼15 W	5.636 Mp/s 9.35 W	0.878 Mp/s ∼15W
Random (2048 x 2048)	3.844 Mp/s 9.50 W	5.478 Mp/s ∼15 W	4.32 Mp/s 9.76 W	7.302 Mp/s ∼15 W	17.047 Mp/s 11.52 W	0.876 Mp/s ∼15 W
Black (2048 x2048)	3.75 Mp/s 9.35 W	6.108 Mp/s ∼15 W	4.66 Mp/s 9.62 W	7.866 Mp/s ∼15 W	16.78 Mp/s 10.28 W	0.86 Mp/s ∼15 W
Random (4096 x 4096)	3.308 Mp/s 9.43 W	5.398 Mp/s ∼15 W	4.14 Mp/s 9.70W	6.014 Mp/s ∼15 W	30.642 Mp/s 13.08 W	0.88 Mp/s ∼15 W
Black (4096 x4096)	3.24 Mp/s 9.24 W	5.912 Mp/s ∼15 W	4.21 Mp/s 9.88 W	6.482 Mp/s ∼15 W	32.43 Mp/s 12.54 W	0.882 Mp/s ∼15 W

Table 2: Performance in MPixels/s and average maximum power execution for CCSDS122 with various images and sizes

expensive transfers, so the CPU sequential version is faster than the GPU one in the AMD platform.

Compared to a state-of-the-art space qualified ASIC implementation [14], which matches the space requirement of 60 MPixels/s compression rate, we obtain half of its performance with double power consumption on the Xavier GPU implementation. However, [14] only supports pixel ranges up to 16 bit, while our evaluation has been performed using 32 bit arithmetic and it is configurable to use up to 64 bits per pixel. This illustrates a significant difference between software implementations and ASIC hardware implementations, that can be easily extended for the requirements of future missions, which will use higher resolutions and larger dynamic ranges per pixel. Moreover, note that the highest performance in our implementations is achieved with larger images, regardless of whether they are random or uniform. This means that our approach will benefit from the larger image sizes which will be used in future missions.

4 Conclusions and Future Work

In this on-going research paper we presented our work on the CPU and GPU parallelization for CCSDS 121 and 122.We have shown that although the parallelization of compression algorithms is challenging, it is possible to obtain significant speedups with their CPU and GPU parallelization, as our results on two embedded GPU platforms show. In fact, our obtained results are very close to the requirements of existing space missions both in terms of performance as well as in power consumption. Moreover, they are competitive with existing space processors.

As a future work we intend to finish our parallel implementations on the CCSDS 123.0-B-1, since it shares several common blocks with the CCSDS 121.0-B-3. Moreover, we would like to further investigate the issue of the low performance of the AMD GPU despite the fact that its characteristics are similar to the NVIDIA one. A possible reason is the custom GPU driver we are using, so we will explore other possibilities to confirm this fact or rule it out.

5 Acknowledgments

This work was funded by the Ministerio de Ciencia e Innovacion - Agencia Estatal de Investigacion (PID2019-107255GB-C21/AEI/10.13039/501100011033 and IJC-2020-045931-I) and partially supported by the European Space Agency (ESA) through the GPU4S (GPU for Space) activity and the HiPEAC Network of Excellence.

References

[1] L. Kosmidis, I. Rodriguez-Ferrandez, A. Jover-Alvarez, S. Alcaide, J. Lachaize, A. C. O. Notebaert, and D. Steenari, “GPU4S: Major Project Outcomes, Lessons Learnt and Way Forward,” in *Design, Automation and Test in Europe Conference and Exhibition, (DATE)*, 2021.

[2] L. Kosmidis, I. Rodriguez, A. Jover, S. Alcaide, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, “GPU4S: Embedded GPUs in Space - Latest Project Updates,” *Elsevier Microprocessors and Microsystems*, vol. 77, Sept 2020.

[3] CCSDS The Consultative Committee for Space Data Systems, *CCSDS 121.0-B-3, Lossless Data Compression*. CCSDS Blue Book, 2020. https://public.ccsds.org/Pubs/121x0b3.pdf.

[4] CCSDS The Consultative Committee for Space Data Systems, *CCSDS 122.0-B-2, Image Data Compression*. CCSDS Blue Book, 2017. https://public.ccsds.org/Pubs/122x0b2.pdf.

[5] ESA, “Obpmark (on-board processing benchmarks),” 2021. http://www.obpmark.org.

[6] Powell, Wesley and Campola, Michael and Sheets, Teresa and Davidson, Abigail and Welsh, Sebastian, “Commercial Off-The-Shelf GPU Qualification for Space Applications,” tech. rep., NASA, 2018.

[7] L. Kosmidis, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, “GPU4S: Embedded GPUs in Space,” in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pp. 399–405, Aug 2019.

[8] Mr. Nan Li, Mr. Aimin Xiao, Mr. Mengxi Yu, Dr. Jianquan Zhang, Dr. Wenbo Dong, “Application of GPU on-orbit and Self-adaptive Scheduling by its Internal Thermal Sensor,” in *International Astronautical Congress (IAC)*, 2018.

[9] F. C. Bruhn, N. Tsog, F. Kunkel, O. Flordal, and I. Troxel, “Enabling Radiation Tolerant Heterogeneous GPU-based Onboard Data Processing in Space ,” *CEAS Space Journal*, vol. 12, pp. 551–564, June 2020.

[10] D. Luchena, V. Schiattarella, D. Spiller, M. Moriani, and F. Curti, “A new complementary multi-core data processor for space applications,” 10 2018.

[11] Unibap AB and Mälardalen University, “"Bruhnspace ROCm project for AMD APUs",” 2020. https://bruhnspace.com/en/bruhnspace-rocm-for-amd-apus/.

[12] U. de Las Palmas de Gran Canaria, “Expro+ esa ao/1-8032/14/nl/ak ccsds lossless compression ip-core space applications,” tech. rep., Universidad de Las Palmas de Gran Canaria, 2017.

[13] N. P. Project, “False color image of the area surrounding yogi, nasa mars pa,” jun 1998.

[14] J.-L. Poupat, “Cwicom & coreci: Towards a highly integrated & innovative image compression unit,” *ESASP*, vol. 694, p. 35, 2011.

Fine-Grained Runtime Monitoring of Real-Time Embedded Systems

Zineb Boukili, Hai Nam Tran, Alain Plantec
Univ. Brest, Lab-STICC, CNRS, UMR 6285, Brest, France; email: {firstname.lastname}@univ-brest.fr

Abstract

Dynamically ensuring the correctness of the functional behavior of a real-time embedded system is tedious, especially in the autonomous domain. Even though the current real-time task model provides sufficient information to perform basic schedulability tests, it is inadequate to be used in runtime monitoring to assert and guarantee the correctness of a system under hardware/-software malfunctions or malicious cyber attacks. In this article, we present a runtime monitoring approach based on a fine-grained model of real-time tasks.

1 Introduction

Real-time systems (RTES) are evolving rapidly in complexity with higher demands in schedulability, safety, and security. In order to satisfy these demands, runtime monitoring is required as a correctness verification procedure to assess the system's state against a previously defined specification. The real-time task model presented in the seminal work of [1] has been widely implemented in real-time scheduling analysis. Even though this model gives sufficient information to perform basic schedulability tests, it is inadequate to enforce a dynamic control to assert the normal operation of a system under hardware/software malfunctions or malicious cyber attacks.

To cope with the problem, we propose a runtime monitoring approach based on a fine-grained model of real-time tasks. For a given task, an execution profile is constructed by exploiting the information obtained from static timing analysis and scheduling simulation tools. Then, verifications are done at runtime, by monitoring the timing properties of tasks and comparing them with the predefined execution profile.

2 Approach

The proposed runtime monitoring approach consists of building the execution profile, implementing monitoring infrastructure, and establishing runtime verification criteria.

Execution profile: The model consists of control flow graphs (CFG) of tasks with the execution time of each basic block. This information is obtained by using static timing analysis tools such as OTAWA [2]. It also uses scheduling simulation results of the tasks over the feasibility interval, which are obtained by a real-time scheduling simulator such as *Cheddar* [3].

Monitoring infrastructure: The first prototype of the monitoring unit is implemented as a software process running on a dedicated core. For a task, a subset of basic blocks is chosen as monitored program points that sent signals to the monitors by instrumenting their code with interprocess communication mechanisms. This approach is known to be time-consuming. In future work, we aim to reduce the overhead by employing a hardware monitor.

Runtime verification criteria: A subset of basic blocks, called monitored program points, is instrumented by adding instructions sending a signal to the monitoring unit. The monitor receives the information with regard to the timing properties of each program point. Verification theory for an individual task is based on three criteria: static timing properties, order of events, and presence of events:

1. A program point p can be executed only between the best-case execution time of the CFG part including nodes that are connected to p but not p itself and the worst-case execution time of the same part including p. Nevertheless, conditional and loop statements are challenging because they make the verdict more difficult. Upperbounds and lowerbounds can be computed by applying graph theory to deduce the shortest and the longest paths.
2. The monitored program points appear in the predefined order in the control flow graph.
3. A program point is on a must executed path, it must appear in the monitored log.

Finally, the monitored events must also conform to the scheduling simulation results provided by the simulator.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[2] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat, "Otawa: An open toolbox for adaptive wcet analysis," in *IFIP International Workshop on Software Technolgies for Embedded and Ubiquitous Systems*, pp. 35–46, Springer, 2010.

[3] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *Proceedings of the 2004 annual ACM SIGAda international conference on Ada*, 2004.

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

Ada-Deutschland

Dr. Hubert B. Keller CEO
ci-tec GmbH
Beuthener Str. 16
76139 Karlsruhe
Germany
+491712075269
Email: h.keller@ci-tec.de
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. Julio Medina
Facultad de Ciencias
Universidad de Cantabria
Avda. de los Castros s/n
39005 Santander
Spain
Phone: +34-942-201477
Email: julio.medina@unican.es
URL: www.adaspain.org

Ada-Switzerland

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch

Ada-Europe Sponsors

27 Rue Rasson
B-1030 Brussels
Belgium
Contact:Ludovic Brenta
ludovic@ludovic-brenta.org

46 Rue d'Amsterdam
F-75009 Paris
France
Contact: Jamie Ayre
sales@adacore.com
www.adacore.com

2 Rue Docteur Lombard
92441 Issy-les-Moulineaux Cedex
France
Contact: Jean-Pierre Rosen
rosen@adalog.fr
www.adalog.fr/en/

22 St. Lawrence Street
Southgate, Bath BA1 1AN
United Kingdom
www.capgemini.com
.
.

Jacob Bontiusplaats 9
1018 LL Amsterdam
The Netherlands
Contact: Wido te Brake
wido.tebrake@deepbluecap.com
www.deepbluecap.com

24 Quai de la Douane
29200 Brest, Brittany
France
Contact: Pierre Dissaux
pierre.dissaux@ellidiss.com
www.ellidiss.com

In der Reiss 5
D-79232 March-Buchheim
Germany
Contact: Frank Piron
info@konad.de
www.konad.de

3271 Valley Centre Drive,Suite 300
San Diego, CA 92069
USA
Contact: Shawn Fanning
sfanning@ptc.com
www.ptc.com/developer-tools

Signal Business Centre
2 Innotec Drive, Bangor
North Down BT19 7PD
Northern Ireland, UK
enquiries@sysada.co.uk
www.sysada.co.uk

1090 Rue René Descartes
13100 Aix en Provence
France
Contact: Patricia Langle
patricia.langle@systerel.fr
www.systerel.fr/en/

Tiirasaarentie 32
FI 00200 Helsinki
Finland
Contact: Niklas Holsti
niklas.holsti@tidorum.fi
www.tidorum.fi

Corso Sempione 68
20154 Milano
Italy
Contact: Massimo Bombino
massimo.bombino@vector.com
www.vector.com

Beckengässchen 1
8200 Schaffhausen
Switzerland
Contact: Ahlan Marriott
admin@white-elephant.ch
www.white-elephant.ch


Ada
europe