Ph.D proposal **ADT4HPC**
High Performance Code Generation for Abstract Data Types

Gabriel Radanne – Inria/LIP,
Laure Gonnord – Université Lyon1/LIP

2021

1 Context

In the last few decades, Algebraic Data Types (ADT) have emerged as an incredibly effective tool to model and manipulate data for programming. Algebraic Data Types are the combination of "Product types", which correspond to records, and "Sum types", which correspond to tagged unions, an extension of traditional enumerations. Algebraic Data Types are also provided with "pattern matching", an extension of `switch` which allow to deconstruct complex values conveniently and safely.

Combined, these features offers numerous advantages:

- Model data in a way that is close to the programmer's intuition, abstracting away the details of the memory representation of said data.
- Safely handle the data by ensuring via pattern-matching that its manipulation is well typed, exhaustive, and non-redundant.
- Optimize manipulation of the data thanks to the presence of richer constructs understood by the compiler.

Initially mostly present in functional programming languages such as OCaml and Haskell, Algebraic Data Types are now present in languages with different paradigms such as web programming with Typescript, Object oriented programming with Scala (and soon Java[1]!) and imperative programming with Rust.

Unfortunately, these features are so far seldom used much in High Performance Computing, notably in languages like C and C++. One of the reason is that, while they provide great convenience and safety, Algebraic Data Types tend to abstract away the exact details of how data is represented in memory. These fine details are exactly the control required by low-level programmers to achieve high performances.

A first step: the example of Rust Rust made a first contribution in this space by providing dedicated optimizations to the representation of ADTs. Let us take the example of the `Option` algebraic data type, which indicates that a value can be present (the `Some` case) or not (the `None` case).

```
enum Option<T> { // Optional values of type T
  Some(T), // Some value of type T
  None, // No value
}
```

The type `Option<Int64>` represents an optional 64-bit integer and has a memory representation using up to two words: one word to distinguish between the two constructors and one word containing the integer. We now consider the type `Option<&T>` of optional pointers to a type `T`. Like machine integers, pointers occupy one word. However null pointers are forbidden in Rust, which means the value 0 is never used. Therefor, an optional pointer can use it to distinguish the `None` constructor which allow Rust to represent values of type `Option<&T>` with only one word. This recovers the efficiency of null pointers, without loosing safety. This concept is also used for other similar types such as file handles (for which -1 is forbidden).

While this optimizations provide huge-gains in the context or Rust, it is generally restricted to ADTs with very simple structures, such as `Option`. Generalizing this optimizations to arbitrary ADTs require developing new dedicated analysis and optimizations tailor-made for this use-case.

[1] `https://openjdk.java.net/jeps/360`

LIP – UMR CNRS / ENS Lyon / UCB Lyon 1 / INRIA – 69007 Lyon
E-mail : Gabriel.Radanne@inria.fr and Laure.Gonnord@ens-lyon.fr

2 Objective and program of the PhD

The goal of this PhD is to investigate extensions of algebraic data types outside of the usual constraints of functional garbage-collected programming languages, specifically targeted at High Performance Computing. The Phd will explore internal memory optimizations as well as compiler optimizations opportunities for HPC languages.

The work would draw inspirations from several sources:

- The extensive literature on Algebraic Data Types, in particular their typing, expressivity and safety considerations [3, 6, 8, 9] but also how to optimize pattern-matching [7].
- The work on "niches" in the context of the Rust compiler, which provided some limited memory optimizations for Algebraic Data Types and give an early hint as to what is possible.
- The work on succinct and cache-aware data-structures, notably trees, which explores how to provide heavily hand-optimized representation for complex data ([1]).
- The first step toward more "HPC-compilers oriented" tree optimizations based on array-based layouts (Tarries) explored in the context of the Phd of Paul Iannetta ([4], another paper submitted).

Such extensions could in particular apply to lower-level languages like Rust or C++, but also inform new developments to provide better control in otherwise higher-level languages, like OCaml and Haskell.

Program

- In the first year, the Phd student will extensively study the above cited sources of inspiration under the point of view of low-level *memory optimization*. Then she will propose a general framework for designing memory representations of general ADTs. Milestones will include non recursive sum types and simple recursive types (with indirections) such as trees. A first proposition for compiling pattern-matching will be designed and compared to previous solutions.
- The second year will be devoted to the construction of a first prototype to demonstrate the pertinence of the previous propositions. The front-end might be a "toy domain specific language" with pattern matching but it will be designed having full recursive ADT support in mind. The validation will be:
  - On expressivity: we will use examples coming from the C HPC community (hand-crafted examples).
  - On performance: we will use benchmarks of the Rust/Ocaml community (with restrictive ADTs shapes compared to the scope of our study).
- For the rest of the Phd, and depending of the current results/ status of the previous activities, the student might study:
  - Recursive ADT structures with no indirection (linearization of trees, for example);
  - In place operations for ADTs;
  - More type constructions (arrays, dictionaries);
  - Application to succinct data structures [2, 5, 10].

2.1 Location and Supervising

The PhD will take place in LIP, Lyon, France in the CASH team [2] It will be supervised by:

- Gabriel Radanne, Inria researcher (70 % on this PhD). No current PhD advising or supervising. Gabriel Radanne has a large experience in the design and implementation of statically typed programming languages, both high and low level. He is also involved in the OCaml ecosystem at large, and the development of the OCaml type-checker in particular, which is one of the most mature language featuring ADTs.
- Laure Gonnord, **habilitated** assistant professor at Lyon 1 (30 % on this PhD). Currently co-advising two third-year PhD students (at 70% each) ending fall 2021. Laure Gonnord has many experiences in static analysis applied to compilation in general, and HPC kernels in particular. She has designed analyses that especially target the LLVM compiler's intermediate representation and has strong knowledge of the polyhedral model framework for optimizing intensive computation kernels. The has supervised the Phd work of Paul Iannetta which has partly inspired this new Phd project

Both Gabriel Radanne and Laure Gonnord are part of the CASH team, which aims to provide solutions for the end-user developers to fully leverage the capabilities of modern hardware. This can imply targeting new hardware such as FPGA, exploiting parallelism, but also developing new language features that can be better optimized by

[2] `http://www.ens-lyon.fr/LIP/CASH/`

the compiler. As such, this project directly integrates in the team's objectives by improving the languages used to develop high performance applications.

References

[1] Gerth Stølting Brodal, Rolf Fagerberg, and Riko Jacob. Cache oblivious search trees via binary trees of small height. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'02, pages 39–48, USA, 2002. Society for Industrial and Applied Mathematics.

[2] O'Neil Delpratt, Naila Rahman, and Rajeev Raman. Engineering the LOUDS succinct tree representation. In Carme Àlvarez and Maria J. Serna, editors, *Experimental Algorithms, 5th International Workshop, WEA 2006, Cala Galdana, Menorca, Spain, May 24-27, 2006, Proceedings*, volume 4007 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 2006.

[3] Sebastian Graf, Simon Peyton Jones, and Ryan G. Scott. Lower your guards: a compositional pattern-match coverage checker. *Proc. ACM Program. Lang.*, 4(ICFP):107:1–107:30, 2020.

[4] Paul Iannetta, Laure Gonnord, and Lionel Morel. On optimizing scalar self-rebalancing trees. In *COMPAS 2020 - Conférence francophone d'informatique en Parallélisme, Architecture et Système*, Lyon, France, 2020. A long version is also available as a research report under the same name.

[5] Dong Kyue Kim, Joong Chae Na, Ji Eun Kim, and Kunsoo Park. Efficient implementation of rank and select functions for succinct representation. In Sotiris E. Nikoletseas, editor, *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, volume 3503 of *Lecture Notes in Computer Science*, pages 315–327. Springer, 2005.

[6] Luc Maranget. Warnings for pattern matching. *J. Funct. Program.*, 17(3):387–421, 2007.

[7] Luc Maranget. Compiling pattern matching to good decision trees. In Eijiro Sumii, editor, *Proceedings of the ACM Workshop on ML, 2008, Victoria, BC, Canada, September 21, 2008*, pages 35–46. ACM, 2008.

[8] Gabriel Scherer and Didier Rémy. Gadts meet subtyping. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7792 of *Lecture Notes in Computer Science*, pages 554–573. Springer, 2013.

[9] Don Syme, Gregory Neverov, and James Margetson. Extensible pattern matching via a lightweight language extension. In Ralf Hinze and Norman Ramsey, editors, *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP 2007, Freiburg, Germany, October 1-3, 2007*, pages 29–40. ACM, 2007.

[10] Sebastiano Vigna. Broadword implementation of rank/select queries. In Catherine C. McGeoch, editor, *Experimental Algorithms, 7th International Workshop, WEA 2008, Provincetown, MA, USA, May 30-June 1, 2008, Proceedings*, volume 5038 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 2008.