Certified and Optimizing Bit Slicing Compiler

Pierre-Évariste Dagand
pierre-evariste.dagand@lip6.fr
Cnrs – Inria Paris – Lip6

Abstract

This internship offers to design, implement and prove the correctness of a *bit slicing* compiler. It will take place in the Whisper team of Inria Paris – LIP6, located at University Paris 6, and will be supervised by Pierre-Évariste Dagand (Cnrs).

It is common knowledge that a modern computer manipulates 64-bit registers. Most programmers therefore have a deeply ingrained conception of the "atom of computation" being a 64-bit value, which could represent a number or a pointer for example. Software bit slicing [Pornin, 2001], also called *SIMD within a register* (SWAR) [Fisher, 2003], is a programming trick by which a 64-bit register is treated by the programmer as 64 1-bit registers. As a result, bitwise operations – for example, the logical negation of a 64-bit register – behave as a SIMD ("single instruction, multiple data") instruction on 64 1-bit registers: we can exploit bit-level parallelism and therefore increase the throughput of some algorithms. This technique is particularly exploited in cryptography for its improved throughput on some cryptographic primitives [Biham, 1997, Canright, 2005, Azad, 2007] but also for its resistance against timing-attack [Käsper and Schwabe, 2009].

Internship objectives: Writing algorithms in a bit-sliced form is a tedious and error-prone task: in C or in assembly, programmers must implement their bit-level algorithms by manipulating 64 such bits at a time, thus obscuring their initial intent and losing the benefit of automated optimizations.

Following an original proposition by X. Leroy, a first step will be to design and implement a programming language for describing bit sliced algorithms. Due to the inherent bit-level nature of this formalism, we shall reuse concepts and techniques exploited by hardware description languages, such as the synchronous dataflow formalism [Biernacki et al., 2008].

For a student interested in advanced compilation techniques, the next step would be to generate optimized code, efficiency being usually measured in terms of *gate count* [Kwan, 2000]. To this end, one could both use standard techniques of Boolean evaluation [Knuth, 2005] while taking advantage of architecture-specific instructions, such as the Streaming SIMD Extensions (SSE) on Intel machines.

For a student interested in verification techniques, the next step would be to specify the semantics of the description language and prove the correctness of the compiler in the Coq proof assistant [Pierce et al., 2014, Gonthier et al., 2015]. This work will build upon the `ssrbits` library [Blot et al., 2016] developed at LIP6.

Beyond this work at the interface between software and hardware, there is also an opportunity to gain in abstraction by presenting some bit sliced algorithms directly in the world of finite fields [Albrecht, 2012, Lidl and Niederreiter, 1994]. Supporting this abstraction with a high-level language and an optimizing compiler could be pursued as part of a PhD project.

Student's profile: We are looking for a student interested in micro-architectural questions, language design and compilation techniques. The compiler will be implemented in OCaml or Coq. Acquaintance with an interactive theorem prover (Coq, or Isabelle) is welcome. Nonetheless, a motivated student with a strong background in functional programming (OCaml, or Haskell) could certainly learn to use Coq along the way. This work is funded by the Émergence(s) program of the City of Paris, thanks to which we can offer a stipend ("gratification") for the duration of the internship.

References

M. R. Albrecht. The m4rie library for dense linear algebra over small fields with even characteristic. In *International Symposium on Symbolic and Algebraic Computation*, ISSAC '12, pages 28–34, 2012. doi:10.1145/2442829.2442838.

V. Azad. Fast AES decryption. Master's thesis, University of California, 2007. URL `http://hdl.handle.net/10211.9/1224`.

D. Biernacki, J.-L. Colaço, G. Hamon, and M. Pouzet. Clock-directed modular code generation for synchronous data-flow languages. In *Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES '08, pages 121–130, 2008. doi:10.1145/1375657.1375674.

E. Biham. A fast new DES implementation in software. In *Fast Software Encryption*, FSE'97, pages 260–272, 1997. doi:10.1007/BFb0052352.

A. Blot, P. Dagand, and J. Lawall. From sets to bits in coq. In *Functional and Logic Programming*, FLOPS'16, pages 12–28, 2016. doi:10.1007/978-3-319-29604-3_2. URL `https://github.com/ejgallego/ssrbit`.

D. Canright. A very compact S-box for AES. In *International Conference on Cryptographic Hardware and Embedded Systems*, CHES'05, pages 441–455, 2005. doi:10.1007/11545262_32.

R. J. Fisher. *General-purpose Simd Within a Register: Parallel Processing on Consumer Microprocessors.* PhD thesis, West Lafayette, IN, USA, 2003. AAI3108343.

G. Gonthier, A. Mahboubi, and E. Tassi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, Inria Saclay Ile de France, 2015. URL `https://hal.inria.fr/inria-00258384`.

E. Käsper and P. Schwabe. Faster and timing-attack resistant aes-gcm. In *Cryptographic Hardware and Embedded Systems*, CHES'09, pages 1–17, 2009. doi:10.1007/978-3-642-04138-9_1.

D. E. Knuth. *The Art of Computer Programming, Volume 4.* Addison-Wesley Professional, 2005. ISBN 0201853949.

M. Kwan. Reducing the gate count of bitslice des. Cryptology ePrint Archive, Report 2000/051, 2000. URL `http://eprint.iacr.org/2000/051`.

R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge university press, 1994.

B. C. Pierce, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjoberg, and B. Yorgey. *Software Foundations.* Electronic textbook, 2014. URL `http://www.cis.upenn.edu/~bcpierce/sf`.

T. Pornin. *Implantation et optimisation des primitives cryptographiques.* PhD thesis, École Normale Supérieure, 2001. URL `http://www.bolet.org/~pornin/2001-phd-pornin.pdf`.