Usuba,
optimizing and trustworthy bitslicing compiler

Darius Mercadier Pierre-Évariste Dagand
firstname.name@lip6.fr
Sorbonne Université, CNRS, Inria, LIP6,
F-75005 Paris, France

Bitslicing consists in reducing an algorithm to bitwise operations (\texttt{AND}, \texttt{OR}, \texttt{XOR}, \texttt{NOT}, etc.), at which point it can be run with bit-level parallelism, viewing a \(n\)-bits register as \(n\) 1-bit registers, and a bitwise \texttt{AND} as \(n\)-parallel \texttt{AND} operators \cite{biham1997}. Bitslicing is thus able to increase performance by exploiting data-parallelism, while improving security by disabling cache-timing attacks — since a circuit runs in constant time. Bitsliced algorithms heavily benefit from SIMD extensions since their throughput is directly proportional to the size of the registers they use.

However, writing a program in bitsliced form is a tedious and error prone task, which produces a code that is hard to read, maintain, and optimize. To relieve the programmers from the burden of manually writing bitsliced code, we developed \textsc{usuba}, a synchronous dataflow language producing bitsliced \texttt{C} code. The benefits of \textsc{usuba} are threefold. First and by design, any software circuit specified in \textsc{usuba} admits a bitsliced (and therefore efficient) implementation. Second, the informal description of symmetric cryptographical algorithms (by means of pseudo-circuits) directly translates into \textsc{usuba}’s formalism: as a result, one can effectively reason about — through a formal semantics — and then run the specification. Finally, \textsc{usuba} generates optimized \texttt{C} code with SIMD intrinsics, without needing the programmer to write any architecture-specific code. Based on the AES implementation of Käsper and Schwabe \cite{kasper2009}, we designed a general model of bitslicing, called \emph{n}-slicing, which gives the programmer fine-grained control over the structure of the \texttt{C} code generated by \textsc{usuba}.

The codes generated by \textsc{usuba} exhibits similar or slightly lower performances than hand-tuned \texttt{C} code on mainstream ciphers (like DES, AES, Serpent, Chacha20), while being able to be transparently ported on various vector extensions (SSE, AVX, AVX2, AVX-512) through a simple compilation flag.

References
