High-Productivity Stream Programming For High-Performance Systems

Rodric Rabbah, Bill Thies, Michael Gordon, Janis Sermulins, and Saman Amarasinghe
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{rabbah, thies, mgordon, janiss, saman}@csail.mit.edu

Applications that are structured around some notion of a “stream” are increasingly prevalent to common computing practices, and there is evidence that streaming media applications already consume a substantial fraction of the computation cycles on consumer machines [6]. Furthermore, stream processing—of voice and video data—is central to a plethora of embedded systems, including hand-held computers, cell phones, and DSPs. The stream abstraction is also fundamental to high-performance systems such as intelligent software routers, cell phone base stations, and HDTV editing consoles.

Despite the prevalence of these applications, there is surprisingly little language and compiler support for practical, large-scale stream programming. Of course, the notion of a stream as a programming abstraction was established decades ago [1], and a number of special-purpose stream languages exist today (see [8] for a review). Many of these languages and representations are elegant and theoretically sound, but they are often too inflexible to support straightforward development of modern stream applications, or their implementations are too inefficient to use in practice. Consequently, most programmers resort to general-purpose languages such as C or C++ to implement stream programs. Yet there are several reasons why general-purpose languages are inadequate for stream programming. Most notably, they do not provide a natural or intuitive representation of streams, thereby reducing readability, robustness, and programmer productivity. Moreover, because the widespread parallelism and regular communication patterns of data streams are left implicit in general-purpose languages, compilers are not stream-conscious and do not perform stream-specific optimizations. As a result, performance-critical loops are often hand-coded in a low-level assembly language and must be re-implemented for each target architecture. This practice is labor-intensive, error-prone, and very costly.

The StreamIt language and compiler effort at MIT is geared toward boosting programmer-productivity while concomitantly delivering high-performance for a wide array of computing targets. The StreamIt language features several novelties that are essential for large scale program development: the language is modular, parameterizable, malleable and architecture independent. In addition, the language exposes the widespread parallelism and communication patterns that are inherent in many streaming programs, and as a result, the compiler can apply aggressive optimizations that are infeasible to perform when using conventional languages. The StreamIt compiler can deliver high performance codes—with speedups ranging up to 4.5×—for a wide array of computing targets, including embedded and desktop uniprocessors (e.g., StrongARM, IA32, and IA64), tiled and multicore architectures (e.g., Raw [11]), or grid computing systems (e.g., a cluster of workstations interconnected by a high-speed local area network).

Our compilation infrastructure automates several domain-specific optimizations that are well known in the DSP domain. In addition, the compiler includes optimizations for computation reordering, load balancing, layout, and routing; all of which are significantly important in high-performance parallel and distributed computing systems. The compiler can also apply a series of cache-ware optimizations to improve the performance of the memory system and ameliorate the effects of the “memory wall”. Furthermore, we have developed a graphical integrated development environment [4] that provides an elaborate debugging framework to interpret and visually represent streaming computation, including the flow of information in parallel and distributed streaming programs.

We believe StreamIt makes a great stride toward delivering the promise of Moore’s Law to the end-user. The StreamIt language, compiler, and development environment empower software engineers and application developers to productively program streaming system in what is today the near-exclusive domain of experts.

Presentation Outline: The presentation will describe the StreamIt language and its salient features [9, 10]. We will focus on the hierarchical nature of the language, highlighting modularity, malleability, and portability. In addition, the talk will provide an overview of the StreamIt compiler infrastructure, which was released publicly on our website (http://cag.csail.mit.edu/streamit). We will describe the compiler in the context of three research thrusts: automating domain-specific DSP optimizations, targeting distributed communication-exposed architectures, and performing cache-ware optimizations.

First, we will present a set of domain-specific optimizations for linear sections of the stream graph [5, 2]. A computation is linear if each of its outputs can be represented as an affine combination of its inputs (e.g., FIR filters, expanders, compressors, FFTs). The StreamIt compiler recognizes linear computation using a simple dataflow analysis. It then exploits the linear properties to perform algebraic simplification and to translate linear computations into the frequency domain (when profitable). These transformations yield an average speedup of $4.5\times$ on a Pentium 3.

Second, we will describe our backend support for tiled and multicore processors, and distributed computing platforms [3]. We use the MIT Raw architecture as an evaluation vehicle for the former, and a cluster of Pentium 3 processors interconnected with a high-speed network as an evaluation vehicle for the latter. To achieve good performance on these parallel targets, the compiler includes phases for work estimation, load balancing, layout, and communication scheduling. The load balancing stage utilizes a novel dynamic-programming algorithm that can be extended to consider a range of hierarchical cost functions. When targeting a 16-tile Raw machine, the compiler achieves an average speedup of $16\times$ compared to a 1-tile Raw machine, and an average speedup of $9\times$ compared to a Pentium 3. The compiler yields similar performance gains on the Pentium 3 cluster.

Third, we will present several cache aware optimizations to improve instruction and data locality, and improve register allocation and scheduling freedom [7]. The optimizations are founded upon a simple and intuitive model that quantifies the temporal locality of a streaming program. The cache aware optimizations in the StreamIt compiler yield a 249% average speedup (over unoptimized code) for our streaming benchmark suite on a StrongARM 1110 processor. The optimizations also yield a 154% speedup on a Pentium 3 and a 152% speedup on an Itanium 2.

REFERENCES

[1] H. Abelson and G. Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, 1985.

[2] S. Agarwal. Linear state-space analysis and optimization of streamit programs. Master's thesis, MIT CSAIL, August 2004.

[3] M. Gordon, W. Thies, M. Karczmarek, J. Lin, A. S. Meli, A. A. Lamb, C. Leger, J. Wong, H. Hoffmann, D. Maze, and S. Amarasinghe. A Stream Compiler for Communication-Exposed Architectures. In *ASPLOS*, 2002.

[4] K. Kuo, R. Rabbah, and S. Amarasinghe. A productive programming environment for stream computing. In *Workshop on Productivity and Performance in High-End Computing*, San Francisco, CA, Feb 2005.

[5] A. A. Lamb, W. Thies, and S. Amarasinghe. Linear analysis and optimization of stream programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Diego, CA, June 2003.

[6] S. Rixner, W. J. Dally, U. J. Kapani, B. Khailany, A. Lopez-Lagunas, P. R. Mattson, and J. D. Owens. A Bandwidth-Efficient Architecture for Media Processing. In *HPCA*, Dallas, TX, November 1998.

[7] J. Sermulins, W. Thies, R. Rabbah, and S. Amarasinghe. Cache aware optimization of stream programs. In *Languages, Compilers, and Tools for Embedded Systems*, Chicago, June 2005.

[8] R. Stephens. A Survey of Stream Processing. *Acta Informatica*, 34(7), 1997.

[9] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A Language for Streaming Applications. In *Proc. of the Int. Conf. on Compiler Construction (CC)*, 2002.

[10] W. Thies, M. Karczmarek, J. Sermulins, R. Rabbah, and S. Amarasinghe. Teleport messaging for distributed stream programs. In *Symposium on Principles and Practice of Parallel Programming*, Chicago, Illinois, June 2005.

[11] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal. Baring it all to software: Raw machines. *Computer*, 30(9):86–93, 1997.