

Merrimac – Supercomputing with Stream

Extended Abstract for GP² Poster Session

Mattan Erez Jung-Ho Ahn Nuwan Jayasena
Timothy J. Knight Abhishek Das François Labonte Jayanth Gummaraju
William J. Dally Pat Hanrahan Mendel Rosenblum

Stanford University

<http://merrimac.stanford.edu>

We are designing the Merrimac supercomputer, which uses stream processors and a high-radix network to achieve high performance at low cost and low power. The stream architecture matches the capabilities of modern semiconductor technology with compute-intensive parallel applications. We expect Merrimac to offer an order of magnitude or more improvement in performance per unit cost compared to cluster-based scientific computers built from the same underlying semiconductor and packaging technology.

Modern semiconductor technology allows us to place hundreds of functional units on a single chip but provides limited global on-chip and off-chip bandwidths. General purpose processor architectures have not adapted to this change in the capabilities and constraints of the underlying technology, still relying on global on-chip structures for operating a small number of functional units. Graphics processors utilize the inherent parallelism of their target application to achieve much higher performance. However, graphics processors have limited on-chip persistent state and rely more on fairly high off-chip bandwidth than on data reuse to achieve high performance. Stream processors, on the other hand, are fully programmable, have a large number of functional units, and utilize a deep register hierarchy with high local bandwidth to match the bandwidth demands of the functional units with the limited available off-chip bandwidth.

A stream program exposes the large amounts of data parallelism available in some application domains, as well as multiple levels of locality. The Merrimac stream processor then exploits the parallelism to operate a large number of functional units and to hide the long latencies of memory operations. The stream architecture also lowers the required memory and global bandwidth by capturing short term producer-consumer locality, such as the locality present within a function call, in *local register files* (LRF) and long term producer-consumer locality in a large *stream register file* (SRF), potentially raising the application's arithmetic intensity (the ratio of arithmetic to global bandwidth).

A single Merrimac chip is organized as an array of 16 data-parallel compute clusters, a streaming memory system, an integrated network interface, and a scalar processor for running control code. Each compute cluster consists of a set of 4 fully pipelined 64-bit multiply-add (MADD) FPUs, a set of LRFs totaling 768 words per cluster, and a bank of the SRF of 8KWords, or 1MB of SRF for the entire chip. The LRFs are connected to individual arithmetic units over short wires providing very high bandwidth, and the SRF is aligned with clusters of functional units so that it only requires local communication as well. We estimate the Merrimac processor chip to be under 130mm² in a 90nm CMOS process, and

achieve a peak performance of 128GFLOPs at the planned 1GHz operating frequency. A Merrimac system is scalable up to a 2PFLOPs 16,384 processor supercomputer.

Another important piece enabling Merrimac's high performance and efficiency is the streaming memory system. A single stream memory operation transfers an entire stream, which is typically many thousands of words long, between memory and the SRF. Merrimac supports both strided access patterns and gathers/scatters through the use of the stream address generators. The memory system provides high-bandwidth access to a single global address space for up to 16,384 nodes including all scalar and stream execution units. Each Merrimac chip has a 128KWords cache with a bandwidth of 8 words per cycle (64GB/s), and directly interfaces with the node's external DRAM and network. The 2GB of external DRAM is composed of 16 Rambus DRDRAM chips providing a peak bandwidth of 38.4GB/s and roughly 16GB/s, or 2 words per cycle, of random access bandwidth. Remote addresses are translated in hardware to network requests, and single word accesses are made via the interconnection network. The flexibility of the addressing modes, and the single-word remote memory access capability simplifies the software and eliminates the costly pack/unpack routines common to many parallel architectures. The Merrimac memory system also supports floating-point and integer streaming add-and-store operations across multiple nodes at full cache bandwidth.

Merrimac is fully programmable and uses a hierarchical stream compiler, which is being co-developed at Reservoir Labs and Stanford University. The *high-level compiler* (HLC) parses the stream program (expressed in a stream programming language such as Brook) and performs global optimizations designed to maximize arithmetic intensity and parallelism. The HLC *maps* the program to the hardware, allocating streams in the SRF and orchestrating memory operations and kernel execution. The *low-level compiler* performs optimized kernel scheduling to the VLIW clusters and machine code generation. While the compiler is still under development, we have run a number of scientific codes on a cycle-accurate Merrimac node simulator. These applications include a force calculation kernel of the GROMACS molecular dynamics package, a finite element solver, and a multi-grid fluid dynamics code. Initial results are very promising and several applications achieved over 30GFLOPs of sustained performance. Moreover, in all the applications, over 95% of the data was supplied out of the LRF and at most 1.5% was serviced by off-chip memory, validating the ability of the register hierarchy to utilize locality at multiple levels.