**EE482C: Stream Architecture**
**Spring Quarter 2001-2002**
**Project Topics**

The course project will involve investigating an open problem related to stream architecture. While the scope of projects is limited by the five weeks available for completion, the intent is that they should lay the groundwork for a piece of research that is worthy of publication in a leading conference.

The following is a list of possible project topics. It is by no means exclusive. If you have an idea for a project that is not on this list, please suggest it to the course staff.

The timetable for the course project is as follows:

| | |
|---|---|
| 4/30 | Discuss project topics during class meeting, project assigned |
| … | Between 4/30 and 5/9 each group must meet with Professor Dally to discuss their proposed project at least once. Special office hours will be made available for these meetings. |
| 5/7 | Project proposal due |
| 5/14 | Project update – the progress of each group will be discussed during class meeting. |
| … | Between 5/14 and 5/23 each group must meet again with Professor Dally to discuss project progress. Special office hours will again be made available for these meetings. |
| 5/23 | Project review – the progress of each group will be discussed during class meeting. |
| 6/4 | On 6/4 or 6/6, each project group will make an oral presentation of their project |
| 6/6 | Project report due. Details on the report requirements will be in the report assignment. |

## *Suggested Project Topics:*

The suggested topics fall into four broad categories: stream processor organization, stream applications, stream programming issues, and stream compilation.

### Stream Processor Organization

**Time vs. Space Multiplexing:** Investigate the pros and cons of time multiplexed stream processors (like Imagine) and space multiplexed stream processors (like RAW). You should explore the continuum of alternatives between these two extremes. Your investigation may include analytical models of program execution time and load balance as well as simulation studies of architecture alternatives. Ideally you would be able to draw a conclusion as to the optimum approach to take to multiplexing (kernel distribution) in a stream processor.

**Conditional Execution:** Study alternative methods of handling conditionals (e.g., if-then-else, do-while) in stream code. You may consider alternatives to or variations of conditional streams and predication. You may also consider using MIMD clusters rather than SIMD clusters – but make sure to consider the impact of the loss of synchronization that this implies. Evaluate the cost and performance of your proposed solutions.

**Cache Organization:** Investigate alternative cache architectures for a stream processor. You may consider putting caches directly on the clusters, putting a cache between the SRF and the memory, or putting a cache in front of each memory bank. Special cache policies, e.g., for locking data to be modified, or to simplify handling of read-mostly data, may be in order. Caches may be multi-ported or multi-banked. Stream caches are fairly unique in that (as long as they are on the memory side of the SRF) their latency does not matter. They serve solely as a bandwidth multiplier. How does this affect their organization? How is coherence handled in a multi-node stream processor system? What cache line sizes are appropriate for stream processors?

**Memory Architecture:** Stream memory systems have different requirements than conventional ones. They are tuned for high bandwidth and long stream accesses rather than for minimizing latency for short memory operations. Consider how stream memory operations affect the memory architecture. For example, what addressing modes should the memory system support: strided, indexed, bit-reversed, reversed, multi-dimensional, and others. Other things to consider are distribution of streams across multiple nodes, and the fact that scatter/gather operations produce many single record references.

**Aspect Ratio:** Given some number of ALUs in a stream processor (order of 100s), what is the best way to distribute these resources across the three axes of parallelism: data parallelism (more clusters), instruction-level parallelism (more ALUs per cluster), and thread-level parallelism (more independent execution engines)?

**Register File Architecture and Bandwidth Hierarchy:** Develop and evaluate alternative register file architectures for a stream processor. Alternatives to the current LRF and SRF might involve more or less distributed LRFs, more than two levels of register files, SRFs that can be accessed randomly within each *lane*, or randomly to any location.

**Legacy Architectures:** Investigate how stream processing can improve the performance of existing processors – (e.g., a P-4) – for example, by using stream scheduling to make better use of the cache.

## Stream Applications

**Application Study:** Pick an application and study how it can be implemented on a stream processor. Possible applications could be scientific codes (e.g., gene

sequence analysis, radiation transport, hydrodynamics, n-body simulation), signal processing (e.g., multi-user detection, adaptive beam forming), packet processing, data mining, etc…. Your study should estimate the performance that streaming can achieve (compared to more traditional architectures) on your application, critique the ability of a stream architecture and programming system to handle your application, and suggest improvements to the architecture and programming system that would make them more suitable for your application.

**Map Legacy Code to Streams:** Study possible methods to automate the conversion of existing (non-stream) programs to a stream programs. You may want to limit your study to vector codes or even to dense matrix codes to simplify analysis. In this case the goal would be to develop analysis procedures that scan loop nests, extract the kernels and streams, and factor out the memory operations. A successful analysis procedure should generate kernels with enough arithmetic intensity to exploit a bandwidth hierarchy.

## Stream Programming Issues

**Irregular Data Structures:** Investigate methods for executing programs with irregular data structures (e.g., arbitrary graphs) on a stream processor. For example, a stream may consist of a set of vertices each with a variable number of incident edges. Consider how to represent irregular data structures, and how typical irregular programs can be efficiently executed using the bandwidth hierarchy of a stream processor.

**Stream Language Design Issues:** Investigate one or more issues in the design of a stream programming language. For example, is it better for a language to have *retained state* in kernels, as in Kernel-C and StreaMIT? or is it better for kernels to be functional – as in Brook? Is it possible to abstract away the details of the hardware – like the inter-cluster communication in Kernel-C without sacrificing efficiency?

**Multi-Node Programs:** Develop and evaluate methods to efficiently map stream programs over multiple stream processing nodes. You may consider partitioning the data, the program, or both across the nodes. Develop methods for the nodes to coordinate execution and communicate data values. You may also wish to consider dynamic/automatic partitioning and load balancing, as well as making decisions on data replication.

**Variable-Length Streams:** Study methods for efficiently handling variable-length streams in a stream program. One possible option is to make kernels restartable or resumable.

## Stream Compilation

**High-Level Stream Compilation Tools:** Develop processes and tools that help map a high-level stream programming language like Brook to a low-level language – like Stream-C/Kernel-C. Given the limited time available you should pick one step of the mapping process and focus your efforts on that step. Necessary steps include converting conditionals to predication/conditional streams, using persistent state (e.g. scratchpad), automatic strip-mining, inserting intercluster communications.

**Better Stream Compilation Tools:** Investigate better methods of stream compilation. This may include methods for kernel fission and fusion where appropriate, scheduling of kernels and memory operations for minimum execution time, and memory and SRF allocation.

**Improved Communication Scheduling:** There are many limitations of the current implementation of communication scheduling. It does not efficiently handle distributed register files with very sparse communication. Also, it does a very poor job of allocating registers – resulting in register overflows for large kernels. Develop new communication scheduling algorithms that overcome these and other shortcomings of the present approach.