Project Proposals

Lecture #12: Tuesday, 14 May 2002
Lecturer: Students of the class
Scribe: Mehdi Tahoori and Jayanth Gummaraju
Reviewer: Mattan Erez

A total of ten project proposals were discussed in today's class. A detailed description of the proposals are available online in the class web site. In this article, we plan to elaborate on the essence of each project. Each project proposal has been divided into four parts. The first part gives the overall objective/goal of the project. The second part discusses the group's basic approach to accomplish the objective. The third part describes the evaluation methodology of the project. Finally, the fourth part lists some interesting questions/suggestions/challenges that surfaced during the presentations.

1 Project 1: On-chip Support for ILP, DLP, and TLP in an Imagine-like Stream Processor

Overall Objective: To determine the aspect ratio of ILP, DLP, and TLP to obtain an optimum cost (silicon area) to performance ratio in an Imagine-like Stream Processor.

Proposed Approach: For each type of parallelism, modify Imagine simulator such that a maximum attainable limit in performance is reached for a fixed cost. The parallelism is increased in the following ways:

- *ILP:* Increasing/changing the functional units per cluster.
- *DLP:* Increasing the number of arithmetic clusters.
- *TLP:* Instantiate multiple Imagine Processors OR make the clusters MIMD within a single Imagine Processor.

Finally, determine the optimum aspect - the ideal number of resources to be allocated for each type of parallelism for maximum performance.

Evaluation Methodology: Test the different configurations on Imagine using several media benchmarks along with Wavelet and 2D-Convolve applications.

Questions/Suggestions/Challenges: The main challenge of this project lies in setting up and evaluating TLP.

2 Project 2: The Viterbi Algorithm as a Stream Application

Overall Objective: Implement and evaluate the usability of Imagine as a Viterbi Decoder, a popular decoder for convolutionally coded messages.

Proposed Approach: Partition the Viterbi Decoder among multiple Imagine clusters. Implement the StreamC and KernelC versions of the decoder. Schedule the operations to obtain maximum performance.

Evaluation Methodology: Run the decoder on the Imagine simulator. Derive conclusions and make generalizations based on the results.

Questions/Suggestions/Challenges: The biggest challenge in the implementation lies in managing the intermediate data. One of the interesting questions asked was what the estimated FLOP/Byte ratio of this application is. The answer: They hadn't empirically calculated this ratio although they believe that the ratio is fairly high and suitable for Stream Processors.

3 Project 3: Brook Compilation

Overall Objective: To build a compiler framework for meta-compiling Brook to StreamC/KernelC of Imagine Processor.

Proposed Approach: Brook is currently compiled to gcc's IR with annotations. Write a meta-compiler that translates this IR into StreamC. As translating into KernelC is very challenging and potentially time-consuming, hand-code the KernelC part for now. Finally, perform simple optimizations like strip-mining.

Evaluation Methodology: Compile StreamMD (a scientific application on Molecular Dynamics) on this framework to obtain a working and possibly optimized StreamC code on Imagine. Hand-code the kernelC code and test on Imagine.

Questions/Suggestions/Challenges: The main challenges are to handle variable length streams and stream reductions. One of the questions asked was why was StreamC code chosen over Imagine's Macrocode. The answer: The StreamC "compiler" already does double buffering, etc. Moreover, a lot of work has been done (and is being done) to efficiently compile StreamC to Imagine ISA.

4 Project 4: Stream Cache Architectures for Irregular Stream Applications

Overall Objective: To determine an efficient Stream Cache architecture for a given Irregular Stream Application.

Proposed Approach: Implement two kinds of Stream Caches on Imagine:

- Memory System Stream Cache: A cache that is in between Main Memory and SRF.
- Cluster Cache: A cache for every cluster.

Each of these kinds have variants depending on how coherency is implemented (non-coherent, partially coherent), how cache misses are handled, and whether or not add-and-store hardware is supported.

Evaluation Methodology: Code three applications that use irregular streams. Evaluate the different stream cache architectures for each of these applications to determine the most efficient cache architecture.

Questions/Suggestions/Challenges: Some interesting issues involved are modeling cache miss penalties, and handling cache misses when cluster caches are used (as clusters execute in SIMD manner).

5 Project 5: Multi-node Programming

Overall Objective: To evaluate and develop methods for mapping stream programs over multiple stream processing nodes.

Proposed Approach: Two Imagine processors are grouped together to form a single node. Four of these nodes are linked to form a basic multi-node configuration block. The IP packet routing algorithm is implemented on this multi-node configuration in two different ways:

- Parallel execution: Each node works on different dataset.
- Pipeline execution: Each node works on a portion of the algorithm and passes the intermediate results to the next node.

Evaluation Methodology: Execution time of a single-stream processor configuration is compared against that of a multi-node configuration for IP packet routing application.

Questions/Suggestions/Challenges: Several questions were raised about how they planned to divide the application of IP packet routing into multiple tasks. The answer: They would be considering both the space and time multiplexing of the application.

6 Project 6: Improving Unstructured Mesh Application Performance on Stream Architectures

Overall Objective: To efficiently implement unstructured mesh applications, like arbitrary graph neighbors access, on Imagine for improving unstructured streaming algorithms on Imagine architecture.

Proposed Approach:

- Create a data-set that exceeds the SRF size with a core computation representative of an unstructured mesh application.
- Write a C application that will be the base for a correct computation.
- Write an implementation on Imagine as it is using conditional streams to handle the irregularity of neighbors.
- Integrate a cache between the SRF and the memory in ISIM to act as a memory bandwidth multiplier and improve the performance of the imagine implementation.
- Tackle the duplication of elements in the SRF by indexing into the SRF by dividing its banks.

Evaluation Methodology: Comparing execution time and memory bandwidth of each approach using *isim*, the cycle accurate simulator of Imagine.

Questions/Suggestions/Challenges: Some issues will arise like conflicting bank indexing, which they would try to handle by requiring a certain latency between an indexing and the return of the values.

7 Project 7: Compiling/Running Stream Programs on Legacy Architectures

Overall Objective: To optimize performance of some representative stream applications on a chosen legacy architecture assuming :

- Impulse memory controller with software control interface
- Non-blocking L1 and L2 cache (with hardware support for L2 cache prefetch)
- Scalar operations

Proposed Approach:

- Take source code in Brook, transform it into legacy C with software control interface to Impulse memory controller to do gathering/scattering memory accesses.
- Compile the C code into assembly with software prefetching of both L1 and L2 cache overlap with normal computation.
- Schedule this code to hide as much memory latency as possible, through software pipelining, stream scheduling, etc. May also consider strip-mining at C level using the knowledge of cache size.

Evaluation Methodology: Evaluating performance improvements due to this explicit organizing of bandwidth hierarchy using the knowledge of memory access pattern from the source language versus simple implementation.

Questions/Suggestions/Challenges: One of the interesting issues involved is cache and memory name-space considerations.

8 Project 8: Mapping Stream Stencils to Imagine

Overall Objective: To investigate compilation tools from Brook to Stream C for stream stencils focusing on the inter-cluster communication required by stencils.

Proposed Approach: Have a set of general communication mechanisms as well as templates that are optimized for certain stream access patterns. As a first cut, translate simple stencils to KernelC. Secondly, implement a script that automatically selects one of these communication templates depending on the input stream access pattern.

Evaluation Methodology: A good performance metric here would be to see how many cycles it takes to perform the communication required for each template. Then different templates can be ranked based on their communication performance.

Questions/Suggestions/Challenges: One of the interesting questions asked was what kinds of stencils (ie squares/rectangles/circles etc) was expected to be used. The answer: To keep it simple, they planned to implement only square and rectangular stencils, which Brook supports.

9 Project 9: Examine Stream Architectures through Cellular Automata

Overall Objective: To implement cellular automata (Game of Life, etc) in Brook, in order to demonstrate the challenges involved in running memory intensive applications on streaming architectures.

Proposed Approach: Implement Game of Life in Brook. Using this as the basic infrastructure, implement a few other cellular automata. If time permits, look into the general problem of handling irregular grid structures on Imagine. Document a general idea of modeling computation.

Evaluation Methodology: By going through the process of implementing the so called *worst case streaming application*, identify the problems involved in handling memory intensive applications.

Questions/Suggestions/Challenges: One question asked was how he planned to represent the neighbors of cells etc. His answer was that he was in the process of determining an appropriate data structure for this.

10 Project 10: Mapping Vector Codes to Streams

Overall Objective: To develop methods to translate certain forms of simple vector or dense matrix code into snippets of Brook code that is relatively efficient. Writing a simple translator to demonstrate its effectiveness.

Proposed Approach: After reviewing relevant literature, manually translate sample vector/dense matrix codes into the high level stream programming language Brook. With this experience, hope to find a number of translations that are conducive to automation. After formulating the candidate targets and the transformations, generalize these transformations to apply to more general cases. Finally, write a translator that uses these transformations to translates a small subset of the vector programming language into efficient stream code.

Evaluation Methodology: As there is no cycle accurate simulator for Brook, try to implement the stream codes in kernel C and stream C to evaluate execution time.

Questions/Suggestions/Challenges: The big challenge is in the division of a sequence of vector operations into kernels to optimize the performance.