Malleable Architectures for Adaptive Computing MIT9904-04

Progress Report: January 1, 2002—June 30, 2002

Arvind, Larry Rudolph and Srinivas Devadas

Project Overview

Field Programmable Gate Arrays (FPGAs) are being used as building blocks in many different adaptive computing systems. We propose a framework for the synthesis of reconfigurable processors based on existing million-gate FPGAs such as the Xilinx XC4000XV series. The instruction sets of the processors are synthesized prior to running each application so as to significantly improve performance or the power dissipated during the execution of the application. Synthesis of instruction sets is made possible by the development of an architecture exploration system for programmable processors. Further, processor caches can be reconfigured in a dynamic manner so as to improve hit rates for multimedia streaming data. This reconfiguration is made possible by implementing several hardware mechanisms such as column and curious caching into the processor cache.

Neither general-purpose microprocessors nor digital signal processors meet all the needs of intelligent personal devices, multimedia players and recorders, and advanced communication applications. Designing special purpose chips for each application is too expensive to be a feasible solution. It is possible that a large reconfigurable device with appropriate tools and infrastructure may be the solution.

We are investigating a revolutionary technology for designing hardware and firmware from high-level specifications. The approach is to synthesize "malleable" processors, with application specific instruction sets, into million-gate FPGAs. The instruction sets of the processors are tailored to each application so as to significantly improve either the performance or the power dissipated during the execution of the application. Synthesis of instruction sets is made possible by the development of an architecture exploration system for programmable processors. This technology can dramatically reduce the time to market in sectors where the standards are changing too quickly or where functionality evolution is too rapid for traditional hardware design.

Progress Through June 2002

Prefetching has been intensively studied as a way of hiding the ever-increasing memory latency. However, existing hardware/software data prefetching techniques tend to focus on scientific programs, which often exhibit simple array accesses. These techniques are not necessarily applicable to general-purpose programs. Moreover, the existing techniques only consider one application even though real systems execute multiple applications in either time-shared or space-shared fashions.

We have studied aggressive hardware/software prefetching mechanisms targeting general-purpose applications considering time-sharing behavior. The basic idea is to exploit schedulers existing in the operating systems and applications. OS schedulers decide which process will execute next, while applications often have central scheduling code that controls their execution. We modify these schedulers to perform data movement before the scheduled processing of that data.

To evaluate our ideas, we have used IBM's MXT (Memory eXtension Technology) machines. This new technology uses a memory compression scheme to effectively double the size of physical memory. All data in physical memory is kept compressed, strictly managed by a memory controller, and an L3 cache has been introduced to hold the uncompressed contents of that memory. One particular server implementation using this technology employs two processors sharing the L3 cache. IBM has donated one of these machines to our group for research. We have used the second CPU as a prefetch engine to experiment with various prefetching schemes. Preliminary results are encouraging, we will report on these results in the next six months.

Caches are used to improve the average performance of application-specific and general-purpose processors. Caches vary in their organization, size and architecture. Depending on the cache characteristics, application performance may vary dramatically. Caches are valuable resources that have to be managed properly in order to ensure the best possible program performance.

One important aspect to cache design is the choice of the replacement strategy that controls which cache line to evict from the cache when a new line is brought in. The most commonly used replacement strategy is the Least Recently Used (LRU) replacement strategy, where the cache line that was least recently used is evicted. It is known, however, that LRU does not perform well in many situations, including timeshared systems where multiple processes use the same cache and when there is streaming data in applications.

Some cache misses can be avoided using prefetching, i.e., predicting that the processor will make a request for a data block, and bringing the data in before it is accessed by the processor. Prefetching can cause cache pollution, for example when a prematurely prefetched block displaces data in the cache that is in use or will be used in the near future by the processor. Cache pollution can become significant, and cause severe performance degradation when the prefetching method used is too aggressive, i.e., too much data is brought into the cache, or too little of the data brought into the cache is useful. It has been noted that hardware-based prefetching often generates more unnecessary prefetches than software prefetching.

We have addressed the above problems using software-assisted replacement mechanisms that can be used independently or combined with prefetching methods to control cache pollution. For simple aggressive hardware prefetch schemes, performance degradation due to pollution can be almost completely eliminated using our prefetch control mechanisms.

Research Plan for the Next Six Months

We will complete our evaluation of the scheduler-based prefetching strategy on the IBM MXT machine. In particular, we will tune the prefetching strategy for interactive applications such as web-servers and compute-intensive benchmarks.

We will investigate the development of a CAD tool to explore and evaluate memory organizations in embedded systems. Memory in embedded systems can be scratch-pad memory or cache. For most applications, different configurations can give vastly different performances. Therefore, it is important to find the best configuration to optimize each application. We will investigate methods to determine near-optimal configurations.

Another important feature required of embedded system memories is predictability, particularly for real-time applications. Caches, while being transparent, and improving performance, are not as predictable in their behavior as scratch-pad memory. Modified replacement strategies such as those used in our pollution control research can be applied to improve the predictability of an embedded processor cache – we will investigate methods to improve memory system predictability (worst-case performance) as well as average-case performance.

A hallmark of next generation applications will be stream processing. We will evaluate the features outlined above in terms of the stream processing required for applications that need to perform low-power, high-speed voice, video, sensory fusion processing.