Malleable Architectures for Adaptive Computing
MIT 9904-04

Progress Report: July 1, 2000—December 31, 2000

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.

For the past six months, we have focused on the malleable cache aspect of a malleable processor. We developed a methodology to improve the performance of embedded processors running data-intensive applications by managing on-chip memory on an application-specific or task-specific basis. We provide this management ability with several novel hardware mechanism, *column, curious, TLB, caching.*

Column caching provides software with the ability to dynamically partition the cache. Data can be placed within a specified set of cache ``columns'' to avoid conflicts with other cached items. By mapping a column-sized region of memory to its own column, column caching can also provide the same functionality as a dedicated scratchpad

memory including predictability for time-critical parts of a real-time application. Column caching enables the ability to dynamically change the ratio between scratchpad size and cache size for each application, or each task within an application. Thus, software has much finer software control of on-chip memory.

Progress Through December 2000

For the past six months, we have focused on the effectiveness of column caching for streaming applications in the practical context of a speech recognition program. First we obtained memory traces of the speech recognition program and analyzed whether there are streaming data in the traces. We indeed found some address ranges containing streaming accesses, and these were used to perform simulations with column caching. Our experimental results show improvements in cache hit ratios.

In particular, we started with the speech recognition program of the SLS group at MIT/LCS, compiled it for a SPARC system and then converted it to be run under our SimICS simulation system. SimICS is a whole system simulator that includes a simulation of interrupts, operating system functions, and all other machine specific operations. Thus we simulate under real “system” conditions.

SimICS alone does not generate memory traces. We had to use two other tools as well as a modification to SimICS. Finally, we had to deal with a mapping between a 64-bit to 32-bit addressing. After much work, we were able to capture an accurate slice of hundreds of millions memory accesses. We analyzed this memory address trace to uncover long streams. Surprisingly, some of the “streams” are used both as streams and as random access addresses. This was an unexpected result that explains some of the poor caching performance of the speech application.

Some improvements in the cache utilization were obtained with column caching and we expect that with more effort, significant improvements can be had. In the range of 114 to 115 million references, a 16% improvement in the cache miss rate was achieved. Such an improvement immediately translates to lower power requirements and higher performance. We have also noticed an improvement in the “worst-case” performance using column caching. This leads us to conjecture that general purpose processors augmented with column caching can lead to more predictable performance.

Over the past six months, we have also be devising accurate models of cache usage in multitasking environments. We have discovered that there is a critical range where the LRU replacement strategy behaves poorly. The range depends on particular ratios of the number of concurrently executing tasks, their respective time quantums, and the size of the cache. Although current generation DSP and general purpose microprocessors usually do not operate in this poor range, we expect this to happen in the near future as cache sizes grow, the number of streaming applications increases, and the relative time quantums shrink.

Research Plan for the Next Six Months

We have shown a case where column caching can improve cache utilization over a normal cache for streaming data. However, there remain many things to do. The most important thing is to be able to change column

assignment dynamically. We plan to implement this in our cache simulator: hiercache. Then, we need to consider an actual hardware implementation of a column cache. We have evidence that these features of hiercache can be implemented practically. Identifying the address and time range of streaming data is another big problem. Although we can identify them from memory traces, we need to have a mechanism to generate them without memory traces in practical situations.

Applications do not execute in isolation; intelligent applications often have a large number of simultaneously executing applications all competing for the same resource. We expect to devise task scheduling and cache partitioning schemes to make the best use of the expensive and scarce on-chip memory resource.