Malleable Architectures for Adaptive Computing MIT9904-04

Progress Report: January 1, 2000 — June 30, 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.

Progress Through June 2000

Column Cache – Embedded DRAM

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.

We have shown that for many traditional DSP applications the optimal amount of scratch-pad memory is highly application phase dependent. For example, an MPEG application consists of three main phases, with the third phase requiring nearly twice the scratchpad memory as the first. Too little scratchpad memory has a serious performance impact and too much has a economic impact. Column caching enables the optimal amount of scratchpad memory to be dynamically allocated with the leftover usable for traditional caching.

Curious Caching – Improving Data Stream Processing

The L2 cache is augmented with a set of address bounds registers that define regions of curiosity. Whenever there is a bus-write operation to a curious address, the L2 cache will “snarf” that value and place it into the L2 cache in a dedicated partition. The partition size can be dynamically varied via column caching techniques. This technique halves the system bus traffic (bandwidth) for streams, such as audio or video. We have also explored adding a small amount of vector processing capability directly into the L2 cache and have found a speedup of nearly 7 fold for an image understanding application.

TRS – A High-level Description and Synthesis Framework

Synthesizable forms of hardware descriptions have traditionally followed a state-centric paradigm that specifies for each state element in the system, its new state after every clock cycle. A designer must explicitly manage the concurrency in a design by scheduling the exact cycle-by-cycle interaction between the different parts of the system. Such a description, whether schematic or textual (RTL Verilog), gives detailed instructions on how to implement a digital circuit, but the same description cannot be easily correlated to the intended functionality of the circuit. Due to this wide abstraction gap between implementation and functional representations, considerable effort and digital design expertise are required to produce a state-centric description from a functional starting point**.**

Hardware design can be simplified by allowing the direct synthesis of operation-centric functional descriptions that specifies the behavior of a system on an operation-by-operation basis. Each operation is described by a triggering precondition and the operation's effect on the system's state as a whole. Semantically, these operations are interleaved atomically and sequentially during an execution. For example, a microprocessor programmer's manual is an operation-centric description that describes the behavior of a processor on an instruction-by-instruction basis. We developed an architectural description language based on the formalism of Term Rewriting Systems. This language captures complex hardware behavior concisely and precisely. A compiler has been developed that can generate a synthesizable Verilog RTL description from an operation-centric TRS description In a design example based on a 32-bit MIPS integer core, the TRS description is less than one-tenth the size of a hand-coded RTL description. New language features have been added as its use continues with the applications described below.

We have described column caching using a TRS description and have been able to describe fairly complex adaptive behaviors. Although we have not yet settled on a specific architecture, the TRS exercise allows for quick and wide architectural exploration.

TLB Caching – A low power, large cache technology

A low power caching strategy has been devised that dramatically reduces the power-wasteful TAG-table lookup during each cache access. It also permits a significantly larger L1 cache without a performance penalty.

Research Plan for the Next Six Months

During the next six months, we expect to critically evaluate our low-power, column, and curious caching techniques to a variety of application domains.   In particular, we will focus on speech and vision applications. We have already begun to port the speech application to our simulation environment.