CS – 15740 Computer Architecture

MULTIPROCESSORS ON A CHIP

Leon Gu (gu+@cs.cmu.edu)

Dipti Motiani (dipti@cmu.edu)

Presentation Slides
(PDF, PPT)

PAPERS

- K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, C. R. Moore, ″ *Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture*,″ in ISCA, 2003.

- Paramjit S. Oberoi and Gurindar S. Sohi, *Out-of-Order Instruction Fetch using Multiple Sequencers*, The 2002 International Conference on Parallel Processing (ICPP-31), Aug. 18-21, 2002.

INTRODUCTION

The papers listed above cover two different aspects of multiprocessors on a chip. The first paper describes an architecture that can be configured in various ways to exploit instruction, thread and data level parallelisms. The second paper describes a mechanism to achieve high-bandwidth fetch by using multiple narrow fetch units operating in parallel.

PAPER 1

Processors are rapidly becoming bound by wire and memory latencies. Furthermore, existing microprocessors have become increasingly specialized, achieving near 'peak' performance on only a small class of applications. Applications incur large swings in performance based on how well they map to a given design. This is a result of the combination of two trends: diversification of workloads and emergence of chip multiprocessors, for which number and granularity of processors is fixed at processor design time. One strategy to combat processor fragility is to build a heterogeneous chip which contains multiple processing cores each optimized for to run a distinct class of workloads. There are two problems with this approach: increased hardware complexity as there is little design reuse and poor resource utilization.

The other strategy is to build multiple homogenous processors on a die. The hardware must be configurable for efficient execution across broad class of application. This is the principle on which the TRIPS architecture is based. The TRIPS chip consists of processing cores and an array of memory tiles connected by a routing network. Each core consists of an array of homogenous processing execution nodes, a banked Instruction Cache, Data Cache, register file and block control logic. The execution nodes contain an integer ALU, an FPU and a set of reservation stations (effectively instruction queue for the node). The reservation stations of all ALUs in a core combine to form a 3D frame space. Instructions are scheduled in this space.

Some of the resources in the TRIPS architecture can be configured to operate differently depending on the mode (instruction, thread or data parallelism). These resources are called polymorphous resources. These resources and configuration are described briefly below:

1. *Frame space*: For ILP, a group of frames called an A-frame is allocated to each hyperblock. Subsequent hyperblocks are speculated and assigned frames. Hence, multiple instruction execution takes place in parallel. For TLP, the allocation of frames is divided among different threads. For DLP, all frames could be combined to form a super-frame in which large loops are unrolled.
2. *Register File Banks*: The extra hardware registers (invisible to the user) can be used in different ways such as speculating for a single thread for ILP or speculating with multi-threading for TLP.
3. *Block Sequencing Control*: The TRIPS architecture is fundamentally block-oriented. In all modes of operation, programs are partitioned into hyperblocks by the compiler. The policies that govern allocating and de-allocating from the frame-space, can differ for different modes of operation. Examples are allocating the next predicted hyperblock for ILP, choosing between hyperblocks of competing threads in TLP and keeping the same hyperblock in the frame-space for repeated executing in DLP.
4. *Memory Tiles*: The memory tiles close to the processor can for a special high-bandwidth interface for applications containing DLP.

A major challenge for polymorphous systems is designing the interfaces

between software and re-configurable hardware and determining when re-configuration should be initiated (statically or dynamically by learning the application requirements). Also, with this architecture the design of the OS and compiler would become considerably complicated.

PAPER 2

Fetching large blocks of contiguous instructions is inefficient for modern out-or-order processors. Trace caches fetch blocks of non-contiguous instructions, but do not use cache space efficiently. Furthermore, an entire trace can be fetched in a cycle, but all the instructions cannot be executed immediately because of dependencies. There is more parallelism between instructions of different traces than between instructions within a trace. The paper describes a mechanism that fetches small blocks of instructions from multiple points in a program (out-of-order instruction fetch).

In this architecture, the fetch unit is replicated and fetched instructions are placed in trace buffers rather than directly into the IFQ. Each trace buffer is a FIFO queue of instructions. Each buffer contains a complete trace and the buffers are linked by next-trace pointers. The IFQ fetches instructions in-order from the oldest trace buffer. The fetch unit fetches instructions sequentially and places them in available trace buffers. A new trace buffer is allocated if a new trace is predicted or a trace being fetched by a fetch unit ends. Prediction is done at the level of traces by a trace-predictor. The instruction cache is banked to allow multiple simultaneous access.

As a trace is read from the trace buffers into the IFQ, the trace is deleted. Reusing the traces in the trace buffers gives a significant performance improvement. Without trace reuse, the performance of multiple sequencers (MS) is just as good as the trace cache (but with improvement in storage efficiency and extra instructions fetched), but with trace reuse, even the performance of MS is better than trace cache.

In this mechanism, instructions are fetched out-of-order, but issued only in-order (the IFQ fetches instructions from the oldest trace, going to the next one in-order). To allow out-of-order execution, instruction renaming must be employed. This technique is described in a later paper.

MS gets the best of both: fetch bandwidth of a trace cache and storage efficiency of an instruction cache. Also, MS is more tolerant to cache-misses for small cache sizes as compared to a trace cache. This is because while handling a miss, other instructions can be fetched.

REFERECES

- http://www.cs.utexas.edu/users/cart/trips/