Automatic Phase Detection and Adaptation for an Asymmetric Multi-Core Environment

Matthew Beckler, Vinod Chandrasekaran
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213
{mbeckler, vinodc}@cmu.edu

Introduction and Background

- Over the past 10 years, as power consumption has become the limiting factor in further scaling processor frequency, CPU vendors have started to use new transistors as extra cores, to facilitate parallel computing.
- Historically, all chip multi-processors (CMP) have been symmetric, with all cores exactly the same. This simplifies the design and test phases of production and increases parallel execution capabilities, but serialized sections of a program can become a performance bottleneck.
- With power being a limiting factor for most processor designs, the most power efficient CMP designs consist of low-power, simple cores. Real world applications are difficult to completely parallelize, usually having a serial portion that limits the overall performance.
- Using and architecture with one high-performance core and multiple low-power cores creates an **Asymmetric CMP**. This type of core retains the multi-threaded performance of a symmetric CMP, but provides improved single-thread performance for serialized sections of program execution.

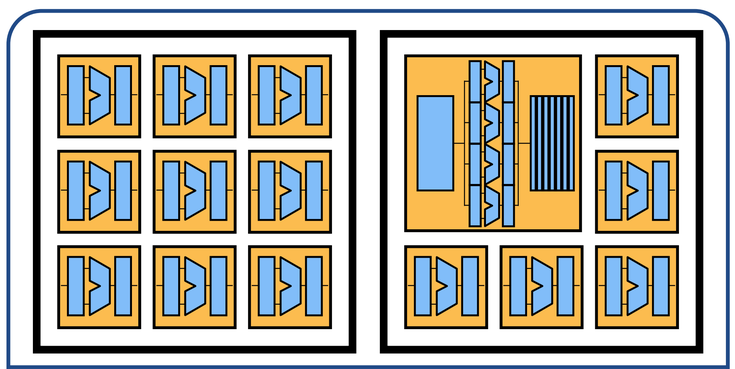

Figure 1: Symmetric and Asymmetric CMPs

Program Phase Detection

- As a program runs, it typically passes through different phases of execution. Each phase may have different performance characteristics and therefore different hardware resource requirements.
- If we can detect significant phase changes, we can adapt by re-allocating suitable processor cores to the current phase. This allows us to optimize processor performance or power consumption on-the-fly.
- Program execution information is collected over a fixed-length sampling interval, measured in terms of instructions executed, and is compared with similar information from the previous interval. A phase change is detected if the difference in value between subsequent intervals exceeds a threshold.

Our Approach:

- We can characterize program phases in terms of the degree of memory- and instruction-level parallelism (MLP and ILP).
- We analyze a sliding window of retired instructions to detect instruction dependencies, giving us a measure of the level of ILP in this interval.
- The degree of MLP can be detected by analyzing the frequency and distribution of L2 cache misses, in order to detect overlapping cache misses.

Previous Work in Phase Detection:

- Sherwood et al. [1] define a Basic Block Vector (BBV) to be a set of counters, each of which counts the number of times a static basic block is entered in a given execution interval. The BBV distance between sampling intervals (*i* and *i-1*) is defined as:

$$\Delta_{i,i-1} = \sum_{j=0}^{\infty} |\mathrm{count}_{i,j} - \mathrm{count}_{i-1,j}|$$

- Smith and Dhodapkar [2] defined a working set as the set of instructions touched over a sampling interval. The distance between the working sets of two adjacent sampling intervals (*i* and *i-1*) is defined as:

$$\Delta_{i,i-1} = \frac{||W_i \cup W_{i-1}|| - ||W_i \cap W_{i-1}||}{||W_i \cup W_{i-1}||}$$

Implementation

- This experiment used a full-system, timing-accurate simulator comprised of Virtutech's Simics [4] functional system simulator and the Flexus architectual models.
- Existing Flexus [5] code supports only symmetric multi-core environments, with all processor cores defined identically. We extended Flexus models to allow unique cores in an asymmetric setup.

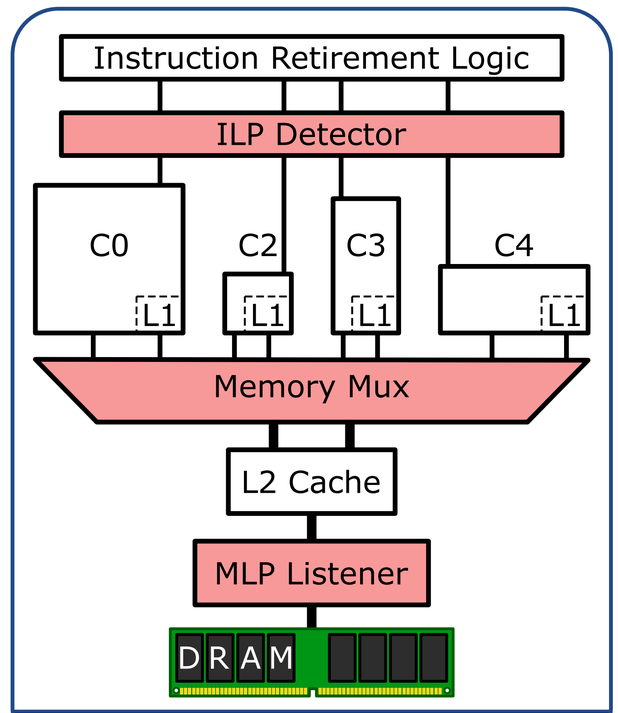

Figure 5: Simulator Block Diagram

New Flexus Modules:

- To measure the amount of memory-level parallelism, we created a new module, the **MLP Listener.** Sitting between the L2 cache and Memory system, it can observe all L2 cache misses.
- To measure the amount of instruction-level parallelism (ILP), a new module was placed between the processor cores and the logic for retiring instructions. This new module, the **ILP Detector**, looks at the dependences between instructions.
- The introduction of asymmetric cores breaks the automatic connections between each core and the memory hierarchy. The **MemoryMux** module sits between the cores and the L2 cache, arbitrating the transfers of data between the individual cores and the shared L2 cache.

Measurement of MLP and ILP:

- Currently, the execution stream is divided into blocks of 1000 instructions. The MLP and ILP are calculated at the end of each block. The ideal number of instructions in this window is still under investigation.

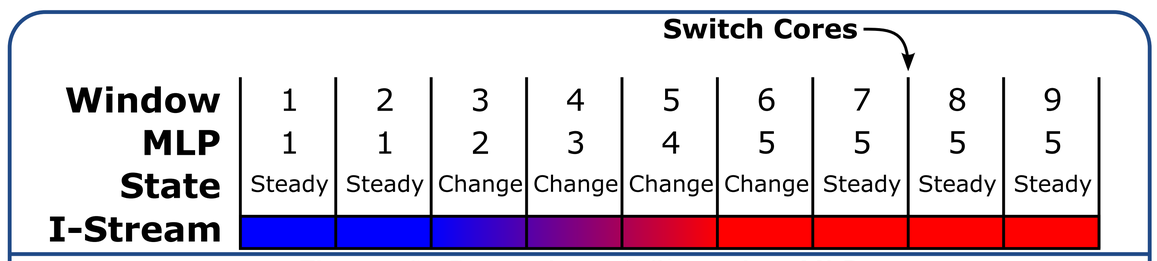

Figure 6: Detection of Changing MLP

- It is very important that each of these metrics is **uArch Independent,** meaning that a given piece of code will receive the same score for MLP/ILP regardless of which core is used for testing and mesurement.

References

1 Timothy Sherwood, Suleyman Sair, and Brad Calder. "Phase Tracking and Prediction", ISCA 2003.

2 Ashutosh Dhodapkar and James E. Smith. "Managing Multi-Configuration Hardware via Dynamic Working Set Analysis", ISCA 2002.

3 Ashutosh Dhodapkar and James E. Smith. "Comparing Program Phase Detection Techniques", MICRO 2003.

4 Virtutech Simics Full-System Simulator, Version 3.0.30, http://www.virtutech.com/

5 SimFlex Project's Flexus Architecture Simulator, Version 3.0.0, http://si2.epfl.ch/~parsacom/projects/simflex/flexus.html

Acknowledgements

We would like to thank Professor Mutlu and the Computer Architecture Teaching Assitants for their help.
We also thank Ben and Steve for their tips and suggestions regarding the Flexus simulator.