Title: Performance Balancing: An Adaptive Helper-Thread Execution for Many-Core Era

Authors: Kenichi Imazato*, Naoto Fukumoto, Koji Inoue, Kazuaki Murakami

*Senior, Department of Electrical Engineering & Computer Science, Kyushu University

1. Concept of Performance Balancing

Conventional CMPs attempt to exploit the thread-level parallelism (TLP) by using all of the cores integrated in a chip. However, this kind of straightforward way does not always achieve the best performance. This is because the memory-wall problem becomes more critical in CMPs, resulting in poor performance in spite of high TLP. To solve this issue, we propose an efficient thread management technique, called performance balancing. We dare to throttle the TLP to execute software prefetchers as helper-threads. When the memory bottleneck causes serious performance degradation, the OS attempts to allocate one or more cores to execute the helper-threads. If the effect of the software prefetching is larger than the negative impact of the TLP throttling, we can improve the CMP performance. Namely, unlike related researches, we attempt to balance the performance of processor and memory at run time.

2. Architectural Support for Software Prefetchers

In our approach, the helper-threads perform prefetching for all of the computing threads. In order to make it possible, we introduce an architectural support, called Miss Status Buffer (MSB), to share the cache-miss information inter cores. Figure 1 shows the CMP architecture targeted in this paper. Each entry of MSB consists of a core-ID, a PC value, and an associated miss address. They can be obtained by snooping the coherence traffic. A helper-thread can be executed on any core because it emulates a hardware prefetcher by referring MSB. By monitoring the processor and memory performance, the OS determines the number of helper-cores and the type of prefetchers. If memory performance is quite worse, OS increases the number of helper -cores.

3. Preliminary Evaluation

We implemented two types of helper-threads, global-stride (GS) and local-stride (LS) software prefetchers [1], and extended the M5 processor simulator[2] to support MSBs. SPLASH2 benchmark set is used in this evaluation. It is assumed that the target CMP has eight in-order cores and a shared 1MB L2 cache. Figure 2 shows the relative speedup over the *BASE* model, in which all cores are used to execute the application-threads. *PB-GS* and *PB-LS* are the models supporting the proposed performance balancing, and execute GS and LS prefetchers as helper-threads, respectively. The value shown over the bars represents the best number of helper-cores for each program. For instance, the value '2' means that six cores are used to execute the application-threads and two cores for helper-threads. In order to evaluate the potential of our approach, the best numbers of helper-cores are analyzed statically. As shown in Figure 2, for some benchmarks, our approach can achieve better performance than the BASE model, 52% in maximum (*Cholesky*). Our ongoing work is to implement an OS scheduler and a performance monitor to adaptively optimize CMP performance.

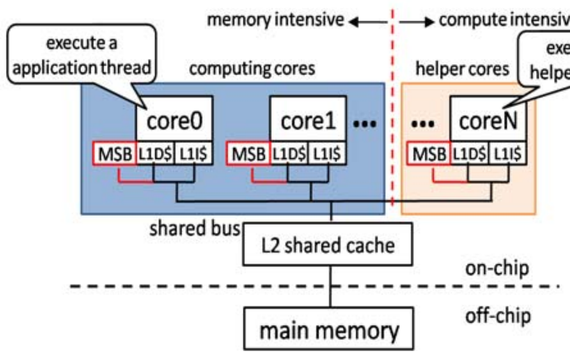

Figure1: Architectural Support

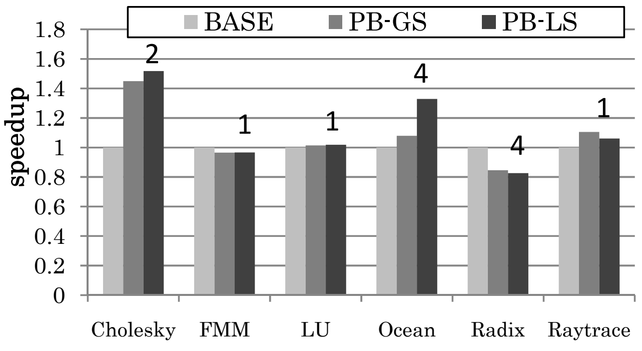

Figure 2: Simulation Result

[1] Ilya Ganusov. et al. Efficient emulation of hardware prefetchers via event-driven helper threading. PACT06, pp. 144-153

[2] Nathan L. Binkert. et al. The m5 simulator: Modeling networked systems. MICRO06, pp. 52-60