Future Generation Computer Systems 30 (2014) 229–241

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Improving execution unit occupancy on SMT-based processors through hardware-aware thread scheduling

Achille Peternier[*], Danilo Ansaloni, Daniele Bonetta, Cesare Pautasso, Walter Binder

University of Lugano (USI), Via G. Buffi 13, 6904 Lugano, Switzerland

HIGHLIGHTS

- We present WorkOver to improve thread-scheduling for better performance.
- We use performance counters to profile integer- and floating-point threads.
- Threads are scheduled according to hardware execution unit availability.
- WorkOver optimizes unit occupancy on AMD Bulldozer and IBM P7 processors.
- We measured up to 20% speedup using Spec CPU and Scimark 2.0.

ARTICLE INFO

Article history:
Received 12 January 2013
Received in revised form
16 May 2013
Accepted 17 June 2013
Available online 1 July 2013

Keywords:
Multicore
Simultaneous multithreading
Workload profiling
Performance
Thread scheduling

ABSTRACT

Modern processor architectures are increasingly complex and heterogeneous, often requiring software solutions tailored to the specific hardware characteristics of each processor model. In this article, we address this problem by targeting two processors featuring Simultaneous MultiThreading (SMT) to improve the occupancy of their internal execution units through a sustained stream of instructions coming from more than one thread. We target the AMD Bulldozer and IBM POWER7 processors as case studies for specific hardware-oriented performance optimizations that increase the variety of instructions sent to each core to maximize the occupancy of all its execution units. WorkOver, presented in this article, improves thread scheduling by increasing the performance of floating point-intensive workloads on Linux-based operating systems. WorkOver is a user-space monitoring tool that automatically identifies FPU-intensive threads and schedules them in a more efficient way without requiring any patches or modifications at the kernel level. Our measurements using standard benchmark suites show that speedups of up to 20% can be achieved by simply allowing WorkOver to monitor applications and schedule their threads, without any modification of the workload.

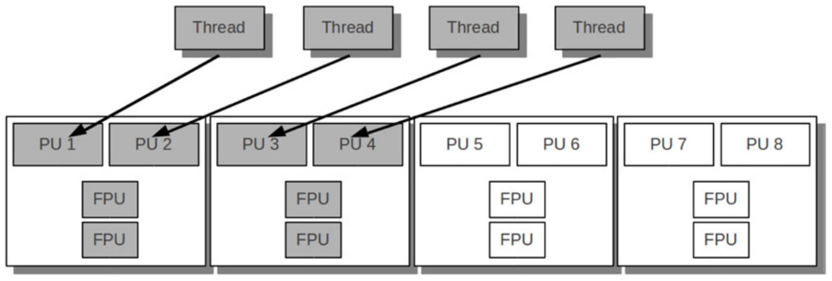
1. Introduction

Since the power wall [1] prevents hardware manufacturers from increasing the processor's clock frequency, modern CPUs embed several cores to increase the computational power through parallelism. Recent trends show that hardware manufacturers are preferring asymmetry and heterogeneity over symmetric and homogeneous designs. Indeed, current state-of-the-art processors have very complex architectures featuring multiple internal components, such as multiple cache levels shared among different cores, Non-Uniform Memory Access (NUMA) [2] controllers and hyperlinks, Simultaneous MultiThreading (SMT) support with several Processing Units (PUs) per core, or ad hoc dedicated units. As a consequence, it is increasingly difficult for software developers to fully exploit the underlying hardware's computational power, as optimal software configurations can vary according to the hardware platform, to the application software architecture, and to the type of workload.

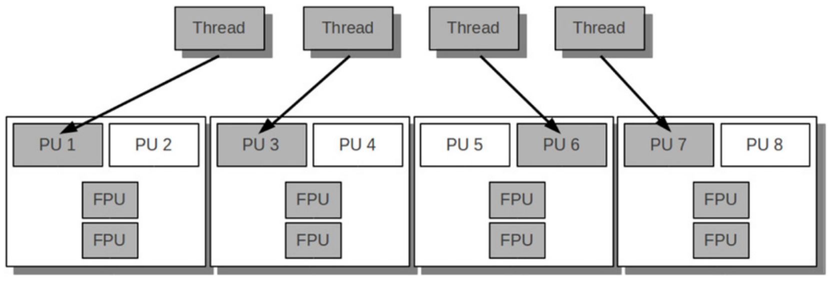
The Operating System (OS) kernel and scheduler try to optimize the performance of applications depending on the available hardware resources. To this end, OS schedulers rely on a limited set of performance indicators (such as the number of cores, CPU time, and memory usage) to drive their optimization strategies. This is often not enough for multithreaded applications running on modern systems, where the complexity and the specific characteristics of the underlying hardware architecture require to use additional information to improve runtime performance through efficient scheduling.

As a case study, in this article we focus on two of these modern architectures and we present a specific, hardware-aware optimization tool based on (1) an automated workload analysis technique

[*] Corresponding author. Tel.: +41 0 76 460 38 37.
E-mail addresses: achille.peternier@gmail.com, achille.peternier@usi.ch (A. Peternier), danilo.ansaloni@usi.ch (D. Ansaloni), daniele.bonetta@usi.ch (D. Bonetta), cesare.pautasso@usi.ch (C. Pautasso), walter.binder@usi.ch (W. Binder).



(a) Inefficient allocation: one thread per PUs without considering the number of FPUs. Only 4 FPUs are used: each thread shares 2 FPUs with another thread.



(b) Optimal allocation: one FPU-intensive thread per core. All the 8 available FPUs are used: each thread uses 2 dedicated FPUs.

Fig. 1. Inefficient vs optimal scattering of 4 floating point-intensive threads on a AMD Bulldozer processor.

relying on a specific set of performance metrics that are currently not used by common OS schedulers, and (2) a hardware-aware optimized scheduler performing scheduling decisions based on hardware resource usage monitoring. Our goal is to use a controller-based approach to profile the workload of multi-threaded and multi-process applications to improve the efficiency of how they share heterogeneous resources.

We focus on two modern micro-architectures that implement very different SMT solutions: the AMD Bulldozer and IBM POWER7 processors. These architectures are good representatives of modern hardware platforms with specific characteristics that cannot easily be exploited by non-hardware-aware approaches. In this context, one of the peculiar characteristics of the Bulldozer architecture is the design of an asymmetric SMT implementation between integer and floating point units, where Floating Point processing Units (FPUs) are shared by two PUs within one same core: two threads may contend for the same FPU units (while integer units are available on a per-PU basis). The IBM POWER7 architecture is based on a more aggressive implementation of SMT, where the instructions coming from up to four threads can be scheduled simultaneously to improve the occupancy of the available execution units on each core. Since each core features two integer and four floating point units, only a proper scheduling of integer- and floating point-intensive threads can take advantage of this improved SMT, otherwise these hardware layouts can have a negative impact on the performance of FPU-intensive workloads.

Our approach is named WorkOver (after Workload Overseer) and corresponds to a Linux daemon that interacts with the OS scheduler to improve the thread scheduling of floating point-intensive workloads on SMT processors by taking into account the way hardware execution units are organized into cores and PUs.

WorkOver runs in user-space and is based on performance metrics commonly available without any modification of the OS kernel and the monitored applications. Our workload profiling approach relies on hardware performance counters to detect which threads make floating point-intensive computations. Our performance optimization is based on improved thread scheduling by pinning the most FPU-intensive threads to PUs of different cores to reduce contention on shared execution units. In this way, WorkOver provides a transparent bottom-up optimization mechanism, based on (1) automatic workload profiling at runtime through performance counters and (2) hardware-aware dynamic allocation of resources. No further intervention is required, neither to modify the running application (the workload) nor to change the OS scheduler. The tool is a system-wide user-mode daemon collecting information and applying optimization policies on the threads spawned by applications (processes) that have been started with a special command.

This article extends our work presented in [3] by generalizing the approach from a specific CPU model to generic SMT processors and by using two completely different hardware architectures and OSs to validate our generalized approach.

2. Motivation and approach

Many scientific applications make heavy use of floating point-intensive computations. Consider a scenario in which a multithreaded application performs floating point-intensive computations with variable intensity in all or a subset of its threads. A common OS scheduler would assign FPU-intensive threads to the available SMT units for execution, as it would do for any other application. The scheduler takes metrics such as CPU time consumption into account. However, prevailing schedulers included in most OS distributions do not consider the way the executed workload is using the hardware resources.

On modern architectures, it makes a significant difference to schedule threads by taking into account the characteristics of the underlying hardware. For simplicity, let us assume that a multithreaded application with 8 running threads has a subset of 4 threads performing FPU-intensive operations. The execution of such application on an AMD Bulldozer four-core processor with 2 PUs on each core (thus seen as a processor with 8 PUs in total) could potentially result in an inefficient use of computing resources. If the OS scheduler scatters the 4 floating point threads to 4 PUs used by two cores (see Fig. 1a), the total number of FPUs used will be 50% less than when the same 4 threads are scheduled one per core (Fig. 1b).

This scenario can be even more detrimental to performance when it happens on a IBM POWER7 four-core processor with 4