Journal of Systems Architecture 60 (2014) 305–314

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

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

An efficient and comprehensive scheduler on Asymmetric Multicore Architecture systems

Jiun-Hung Ding [a], Ya-Ting Chang [a], Zhou-dong Guo [a], Kuan-Ching Li [b], Yeh-Ching Chung [a,*]

[a] Dept. of Computer Science, National Tsing Hua University, Taiwan
[b] Dept. of Computer Science and Information Engineering, Providence University, Taiwan

ARTICLE INFO

Article history:
Available online 29 May 2013

Keywords:
Asymmetric architecture
NUMA architecture
Single-ISA heterogeneous multicore processors
Scheduling

ABSTRACT

Several studies have shown that Asymmetric Multicore Processors (AMPs) systems, which are composed of processors with different hardware characteristics, present better performance and power when compared to homogeneous systems. With Moore's law behavior still lasting, core-count growth creates typical non-uniform memory accesses (NUMA). Existing schedulers assume that the underlying architecture is homogeneous, and as consequence, they may not be well suited for AMP and NUMA systems, since they, respectively, do not properly explore hardware elements asymmetry, while improving memory utilization by avoid multi-processes data starvation. In this paper we propose a new scheduler, namely NUMA-aware Scheduler, to accommodate the next generation of AMP architectures in terms of architecture asymmetry and processes starvation. Experimental results show that the average speedup is 1.36 times faster than default Linux scheduler through evaluation using PARSEC benchmarks, demonstrating that the proposed technique is promising when compared to other prior studies.

1. Introduction

With advances in microprocessor technologies and accelerated with development of multicore, manycore- and embedded systems-related technologies last years, processors evolve to include more processing units – hundreds to thousands of cores – into one single die and widely exploited in High Performance Computing, by harnessing processor architectures in parallel with other technologies and techniques to achieve such high performance.

Asymmetric Multicore Processors (AMPs) system is recently introduced, as composed of processors with different characteristics, e.g., clock speed, cache capacities, power consumption, occupied area and the complexity of execution pipeline, containing or not the same Instruction Set Architecture (ISA), also known as single-ISA heterogeneous multicore [1,3]. Instances of AMP system may contain a few powerful and effective cores and a larger number of cores with slower speed and less power consumptions [1,3]. Many strategies are employed to explore few powerful out-of-order with higher clock speeds and large cache capacities, suitable for executing the throughput oriented applications and single-threaded sequential applications, while for slower but less power-consuming cores, for parallel execution. Such an idea has been considered by major manufacturers as IBM, AMD and Intel, to combine 32- or 64-bit ×86 or Power cores with capable graphics processing units (GPUs) or Synergistic Processor Elements (SPEs) on a single silicon die, e.g., IBM's cell processor [21], AMD's APU [20] and Intel's Larrabee [19]. Prior studies show that the typical AMP system has significant energy benefits and occupies minor die area, yet maximize the power efficiency [1,2,8]. As result, given the core-count growth, access time to memory is variable and depends on the relative location of a processor, which characterizes it as Non-Uniform Memory Access architecture (NUMA) [17]. With rapid growth on the number of cores in computing systems, the amount of memory requests issued by processor cores increases memory starvation.

This limitation on the number of memory accesses decreases the performance of modern multicore systems, and can starve several processors at the same time. In NUMA systems, this problem is settled by providing separate memory for each processor, which is likely to lift the performance when several processors attempt to access same memory. Unfortunately, current OS schedulers assume that the underneath hardware is homogeneous, that is, AMP systems and NUMA architecture are not considered as well as decoupled. Taking as example Linux 2.6 Completely Fair Scheduler (CFS), this scheduler uses a red–black tree implementation to manage the executable processes instead of running queue per processor. The main idea of CFS is to provide processor time to each task fairly. For instance, in a system with n executable processes, each of them should be given $1/n$ process time of a tiny period.

* Corresponding author. Address: Dept. of Computer Science, National Tsing Hua University, No. 101, Section 2, Kuang-Fu Road, Hsinchu 30013, Taiwan. Tel.: +886 3 5742971.
E-mail address: ychung@cs.nthu.edu.tw (Y.-C. Chung).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.05.006

Since the abilities of processors in AMP systems are different, $1/n$ process time in faster and slower processor cores are completely different. Hence, the scheduler should take the AMP architecture into account. In another direction, NUMA architecture can be used to avoid contention of memory accesses between processes, by dividing the memory into multiple nodes, exploring the high-speed interconnections among them, e.g., Intel's Quick Path Interconnect (QPI) and AMD's Hyper Transport (HT). However, given the higher core-count growths and consequent large NUMA architectures formed, combined to the different AMP hardware adaptive design, can provide smaller memory resource contention and avoid data starvation. Again, the scheduler must consider the NUMA architecture in order to get additional benefits from this computer memory design. Based on this tendency, we believe that the AMP and NUMA are essential as the next generation of hand-held devices' architecture. In order to make OS working well with AMP and NUMA, we propose a new scheduler policy, NUMA-aware Scheduler for Asymmetric Multicore Processors, to support AMP and NUMA architectures. Interesting components as target in our proposed scheduler policy are twofold. The former one is Asymmetric-aware schedule policy, where dynamically trigger AMP scheduler to place the suitable processes on the specific type of cores, while the latter one is NUMA-aware schedule policy, in which precisely calculates the current system performance degradation due to resource contention, minimizing the degradation by thread migration and memory management.

The proposed NUMA-aware Scheduler for AMP (Asymmetric Multicore Processors) is implemented in Linux CentOS release 6.0 and evaluated on a 8-core, 32 GB Dell PowerEdge R910 system. Using performance counters, we independently modulated the CPU frequency as a performance asymmetry factor and explored the NUMA memory space to avoid resource contention. Comparing to Linux CFS scheduler and execution of PARSEC benchmarks, the proposed scheduler improves performance by a factor of 1.36×.

The remaining of this paper is organized as follows. In Section 2, some related works are presented, while the overview of NUMA-aware Scheduler for Asymmetric Multicore Processors is given in Section 3. The design of the NUMA-aware Scheduler for Asymmetric Multicore Processors is discussed in Section 4, and evaluation is shown in Section 5. Finally, Section 6 summarizes our findings, as also brings some remarks and topics for future research.

2. Related work

There are several references in literature showing energy benefits of Asymmetric Multicore Architectures [1,2,8]. The research study in [1] showed that this architecture could achieve a large amount of energy reduction with small performance penalty. In order to accommodate the heterogeneity of Asymmetric Multicore Processors, there are several researches [1–9] that discussed scheduling algorithms. Some of them considered the load balancing policy and then implemented an Asymmetric-aware load-balancing [3]. Therefore, the processes' characteristics were not taken into account. Some of them made use of static-time profiling data [4,5], in which could not detect the phase change of a process during runtime. Krumar et al. [1,2] proposed a dynamic core selection based on actual execution performance between different types of cores, and therefore, threads migrate between different cores. Unfortunately, the thread migration overhead is redundant and the cost is high, especially on NUMA architectures. Some of them proposed a dynamic way [6] to implement the scheduler during runtime and profiling data simultaneously, though the periodic profiling and computing are time consuming. Furthermore, the NUMA architecture is not fully considered in such study.

The proposed scheduler based on dynamic computed metric is surprisingly accurate and processes did not have to execute on different type of cores. We made use of hardware counter to gather periodically system's information with tiny overhead and computed the corresponding metric when necessary. Therefore, we minimized the overhead as best as possible and scheduled the processes properly. Hence, modern multicore systems increasingly use the NUMA architecture, and it had been discussed for several years [10–15]. NUMA architectures have benefits, but the system could not learn to profit with proper utilization. Yang et al. [10] analyzed the on-chip interconnect and intra-core bandwidth contention, and then showed the importance of load-balancing between threads. Blagodurov et al. [11] presented a NUMA-aware contention management to reduce the performance degradation; Majo et al. [14] solve the problem by taking both interconnect overhead and cache contention into consideration. In addition, Pusukuri et al. [15] dynamically reduced the performance variation due to NUMA architectures.

3. Proposed scheduler

The purpose of a process scheduling is to optimally sort independent processes according to a given parameter and then execute them. In proposed NUMA-aware Scheduler for AMP, the schedule policy is based on ranking processes according to two metrics, Online AMP Speedup Factor and Resource Contention Degradation Factor, to determine how appropriate they are to be run on certain type of core, the faster core or the slower core, or domain. In the aim of deriving these two metrics for a process, we need a runtime profiler to get ready this information. The AMP Speedup Factor and Contention Degradation Factor are recalculated once the schedule function is invoked.

In order to avoid redundant calculation yet minimize the overhead, the schedule function has to be invoked properly. We implemented two ways to invoke such the schedule function. The former one occurs when the process voluntarily releases the faster core, we have to invoke the scheduler directly for averting from losing greater ability of faster core, while the latter one is a lazy way to trigger the schedule function, invoked when we predict that there will be suitable candidate to run on faster core.

3.1. Framework components

The proposed NUMA-aware AMP scheduler is composed of two components, a runtime profiler and a scheduler. We use OProfile [22] as our system-wide profiler, leveraging the hardware performance counters to profile and analyze the statistic information at low overhead. After a time interval, we dump the profiling data to the data dealer. Once the data dealer receives the profiling data, it will update the records. In case the current situation cause the schedule function invoked, it computes two dynamic metrics: Online AMP Speedup Factor and Contention Degradation Factor. In addition, two important linked lists are maintained, AMP-list and NUMA-list, according to the dynamic values of metrics. In this way, the design method makes the data dealer

Algorithm 1. PROFILER: online profiling mechanism

```
1  Create a new thread for receiving and dealing with the
   online profiling data
2      Repeat profiling until NUMA-aware P-AMP scheduler
   stop
3          Sleep for an OPROFILE_PERIOD amount of time
4            Dump the profiling report
5    End Repeat loop
```