Theoretical Computer Science 550 (2014) 1–20

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Energy-efficient multiprocessor scheduling for flow time and makespan

Hongyang Sun [a,*], Yuxiong He [b], Wen-Jing Hsu [a], Rui Fan [a]

[a] *School of Computer Engineering, Nanyang Technological University, Singapore*
[b] *Microsoft Research, Redmond, WA, USA*

A R T I C L E I N F O

Article history:
Received 28 November 2013
Accepted 6 July 2014
Available online 16 July 2014
Communicated by P. Widmayer

Keywords:
Multiprocessors
Online scheduling
Dynamic speed scaling
Energy-performance tradeoff
Competitive analysis
Total flow time
Makespan

A B S T R A C T

We consider energy-efficient scheduling on multiprocessors, where the speed of each processor can be individually scaled, and a processor consumes power s^α when running at speed s, for $\alpha > 1$. A scheduling algorithm needs to decide at any time both processor allocations and processor speeds for a set of parallel jobs with time-varying parallelism. The objective is to minimize the sum of the total energy consumption and certain performance metric, which in this paper includes total flow time and makespan. For both objectives, we present instantaneous parallelism-clairvoyant (IP-clairvoyant) algorithms that are aware of the instantaneous parallelism of the jobs at any time but not their future characteristics, such as remaining parallelism and work. For total flow time plus energy, we present an $O(1)$-competitive algorithm, which significantly improves upon the best known non-clairvoyant algorithm. In the case of makespan plus energy, we present an $O(\ln^{1-1/\alpha} P)$-competitive algorithm, where P is the total number of processors. We show that this algorithm is asymptotically optimal by providing a matching lower bound. In addition, we study non-clairvoyant scheduling for total flow time plus energy, and present an algorithm that is $O(\ln P)$-competitive for jobs with arbitrary release time and $O(\ln^{1/\alpha} P)$-competitive for jobs with identical release time. Finally, we prove an $\Omega(\ln^{1/\alpha} P)$ lower bound on the competitive ratio of any non-clairvoyant algorithm.

1. Introduction

Energy has been widely recognized as a key consideration in the design of mobile and high-performance computing systems. One popular approach to controlling the energy consumption is by dynamically varying the speeds of the processors, a technique generally known as *dynamic speed scaling* [15,27,48]. Major chip manufacturers, such as Intel, AMD and IBM, have produced chips that enable the operating systems to perform dynamic power management using this technology. It has been observed that, for most CMOS-based processors, the dynamic power consumption satisfies the *cube-root* rule; that is, the power consumption of a processor is proportional to s^3 when it runs at speed s [15,39]. Since the seminal paper by Yao, Demers and Shenker [49], who initiated the theoretical investigation of energy-efficient scheduling, many algorithmic researchers have adopted a more general power function of s^α, where $\alpha > 1$ is called the *power parameter*. As this power function is strictly convex, using dynamic speed scaling can result in a non-linear tradeoff between the energy consumption and the performance, and this has led to many interesting new research problems. One challenging problem concerns how

[*] Corresponding author.
E-mail addresses: sunh0007@ntu.edu.sg (H. Sun), yuxhe@microsoft.com (Y. He), hsu@ntu.edu.sg (W.-J. Hsu), fanrui@ntu.edu.sg (R. Fan).

http://dx.doi.org/10.1016/j.tcs.2014.07.007
0304-3975/© 2014 Elsevier B.V. All rights reserved.

to balance the conflicting objectives of low energy and high performance. The problem has attracted much attention of the algorithmic community and has become an active research topic in recent years. (See [2,32] for two surveys in the field.)

In this paper, we study the problem of scheduling parallel jobs on multiprocessors for the energy-performance tradeoff. We focus on systems with per-processor speed scaling capability; that is, the speed of each processor can be individually scaled [30,50,51]. This kind of architecture has been made possible by the recent advancements in chip design technology, such as the on-chip switching regulators [36,35]. Under this setting, a scheduling algorithm needs to have both a *processor allocation policy*, which determines the number of processors allocated to each job, and a *speed scaling policy*, which determines the speed of each allocated processor. Moreover, we assume that the parallel jobs can have time-varying parallelism in different phases of their executions [22,17,44]. This poses an additional challenge compared to scheduling sequential jobs. In particular, it requires a scheduling algorithm to have dynamic policies in order to respond to the jobs' different resource requirements over time. Badly designed algorithms could either waste a large amount of energy or cause severe execution delays and hence performance degradations.

Our objective is to minimize a linear combination of energy consumption and certain performance metric, which in this paper includes total flow time and makespan. The *flow time* of a job is the duration between its release time and completion, and the *total flow time* is the sum of the flow time of all the jobs in the system. The *makespan* is the largest completion time of the jobs. Both total flow time and makespan are widely used performance metrics: the former measures the average waiting time of all users in the system, and the latter is closely related to the throughput of the system. Although energy and flow time (or makespan) have different units, optimizing a linear combination of the two has a natural interpretation if we consider a user who is willing to spend one unit of energy in order to reduce ρ units of total flow time (or makespan).[1] In fact, minimizing the sum of conflicting objectives has been a common approach in many bi-criteria optimization problems [3,37], and similar metrics have been considered previously in the scheduling literature by combining both performance and the cost of scheduling as a single objective function [47,43,20].

Since Albers and Fujiwara [3] first considered the problem of minimizing total flow time plus energy, many results (e.g., [7,38,37,6,16,17,44,26,4,5]) have been obtained under different online scheduling settings. Some of these results assume that the scheduling algorithm is *clairvoyant*; that is, it gains complete knowledge of all job characteristics immediately upon the job's arrival. Other results are for an arguably more practical *non-clairvoyant* setting, where the scheduler knows nothing about the un-executed portion of a job. Most of these results, however, are only applicable to scheduling sequential jobs. Also, to the best of our knowledge, no previous work has considered minimizing makespan plus energy. The closest result to ours is by Chan, Edmonds and Pruhs [17], who studied non-clairvoyant scheduling for parallel jobs on multiprocessors to minimize total flow time plus energy. In both [17] and our previous work [44], it has been observed that any non-clairvoyant algorithm that allocates a set of uniform-speed processors to a job will perform poorly; in particular, a lower bound of $\Omega(P^{(\alpha-1)/\alpha^2})$ on the competitiveness has been shown for any such algorithm, where P is the total number of processors. The reason is because a non-clairvoyant algorithm may in the worst case allocate a "wrong" number of processors to a job as compared to its parallelism, which will lead to either wasted energy or delayed job execution.

To obtain a better competitive ratio, it turns out that a non-clairvoyant algorithm needs to assign processors of different speeds to a job. To this end, Chan, Edmonds and Pruhs [17] proposed an execution model, in which a job can be simultaneously executed by several groups of processors. The processors within the same group must share the same speed, but different groups can run at different speeds. The execution rate of the job at any time is determined by the group with the fastest speed.[2] They proposed a non-clairvoyant algorithm called MultiLaps, and showed that it is $O(\log P)$-competitive with respect to total flow time plus energy for any set of parallel jobs. They also gave an $\Omega(\log^{1/\alpha} P)$ lower bound on the competitive ratio of any non-clairvoyant algorithm under this execution model.

In this paper, we propose an alternative execution model, under which only one group of processors, possibly with different speeds, can be allocated to a job at any time. The execution rate of the job is determined by the speeds of the fastest processors that can be effectively utilized. This model is based on the assumption that the *maximum utilization policy* [33,9] is employed at the underlying task scheduling level, which always utilizes faster processors before slower ones. Compared to the execution model proposed in [17], our model may be implemented more easily especially for data-parallel jobs with independent and sufficiently long tasks. Our first contribution includes a non-clairvoyant scheduling algorithm and its analysis under this execution model. The following states our results:

- We propose a non-clairvoyant algorithm N-EQUI (Non-uniform Equi-partitioning), and show that it is $O(\ln P)$-competitive with respect to the total flow time plus energy for any set of parallel jobs with arbitrary release time, and $O(\ln^{1/\alpha} P)$-competitive for jobs with identical release time. Moreover, we prove that any non-clairvoyant algorithm is $\Omega(\ln^{1/\alpha} P)$-competitive under our execution model, showing that N-EQUI is asymptotically optimal in the batch-released setting.

Another contribution of this paper is to study a setting that lies between clairvoyance and non-clairvoyance. In this intermediate setting, a scheduling algorithm is allowed to know the available parallelism, or the *instantaneous parallelism (IP)*, of

[1] By scaling the units of time and energy, we can assume without loss of generality that $\rho = 1$.
[2] In practice, this can be implemented by proper checkpointing of the executing program.