

Memory-aware Green Scheduling on Multi-core Processors

Frederic Pinel, Johnatan E. Pecero, Pascal Bouvry
CSC Research Unit
University of Luxembourg
L-1417 Luxembourg
Email: {firstnam.lastname}@uni.lu

Samee U. Khan
NDSU-CIIT GCC Laboratory
North Dakota State University
Fargo, ND 58108-6050, USA
Email: samee.khan@ndsu.edu

Abstract—Contention on shared resources such as cache and main memory slows down the execution of the applications affecting not only application performance but also induces inefficient use of energy. Therefore, in this paper we deal with the contention problem and energy optimization on shared resources multicore-based machines. Our main contribution is a memory-aware resource allocation algorithm that minimize energy consumption by reducing contention conflicts and maximizing performance. We design a heuristic that includes in its objective function the impact of the contention on the application performance. Experimental results emphasize the interest of the provided solution.

Keywords-multi-core processors; memory-aware; green computing; operating systems; contention; scheduling;

I. INTRODUCTION

Design innovations on modern processors allow two or more independent execution cores to be integrated into a single processor, with each core having its own resources. Yet, some interdependencies between cores need be taken into account for optimal performance and energy efficiency. At different levels, cores in a processor share some resources leading to contention if applications running in parallel on the cores compete for the shared resources. Contention can induce not only to degrade the application performance, but also to inefficient use of energy, since cores waiting for a resource to become available dissipate energy without carrying out progress [1].

For example, we have carried out a simple experiment to demonstrate how contention for shared resources can slow down the application performance. In this experiment, we have executed one benchmark application (stream [9]) on a 2.4 GHz Intel Core 2 Duo. This application is main memory bound. First, we have executed the application activating only one core and we measured the time that the application need to complete its execution. It took around 10 seconds. Then, we ran two stream applications in parallel on the two cores sharing a memory domain. The completion time of both applications was around 20 seconds because of the contention. This example clearly shows the negative effects on the performance and energy consumption without any consideration of contention on multi-core processors.

We are interested in the efficient executions of the applications on shared resources based computing machines

with the aim of minimize both the energy consumption and the completion time. Since the resource manager is the component of a system responsible for deciding which application run on the processors simultaneously, resource allocation and scheduling are crucial for performance and energy efficiency. Therefore, our main contribution in this paper is a resource allocation heuristic that take into account contention when scheduling applications. The heuristic is based on the Min-Min resource allocation algorithm [2], [3]. The algorithm consider the multi-objective problem. This algorithms first compute the completion time for each job on each core. The core that has the minimum completion time for each application is selected (it corresponds to the first Min objective in the algorithm and is based on estimated completion times for the applications). Then the application with the overall minimum completion time is selected and affected to the machine or the core (it corresponds to the second Min objective of the algorithm). And this process is repeated. We have modified the first function of the Min-Min algorithm by adding a parameter that take into account the performance degradation on the completion time for the application when it is in memory contention conflict with another application that is in execution.

This paper is organized as follows. In Section II, we will describe the problem. The proposed approach is defined in Section III. Section IV and V provide some results and conclude the paper.

II. PROBLEM STATEMENT

We consider computing architectures with a set P of M heterogeneous processor packages each package containing a set of m homogeneous/heterogeneous cores. The cores in general will be heterogeneous in time and energy requirements. These cores share last level cache and memory. One example of this kind of architecture could be machines with large shared memory having two 2.5 GHz Intel Xeon E5000 processor package series of two or four cores each. Each core may only execute one task at a time (i.e., no multi-tasking). Furthermore, we assume that processor packages are equipped with DVS features.

We assume a set of independent applications. They have to be individually processed by a single resource (*non-*

preemptive mode). The applications could specify hardware and/or software requirements over resources. These applications could be memory bounded and/or cpu intensive. For each application, we assume that an estimated time to compute (ETC) on each core has been provided. This ETC is estimated without consideration of contention. However, in our model we provide the resource allocation heuristic with a mechanism that penalize the objective function in the case of contention. An the final assignation of applications to processing elements are done taking into account this situation. More precisely, assuming that the computing time needed to perform a task is known (assumption that is usually made in the literature [3], [5], [6]), we use the ETC model by Braun et al. [3] to formalize the instance definition of the problem as follows:

- A *number* of independent (user/application) *tasks* to be scheduled.
- A *number* of heterogeneous *machine/cores* candidates to participate in the planning.
- The *workload* of each application (in millions of instructions).
- The *computing capacity* of each machine/core (in *mips*).
- Ready time $ready_m$ indicates when core m will have finished the previously assigned tasks.
- The Expected Time to Compute (*ETC*) matrix ($nb_tasks \times nb_machines$) in which $ETC[t][m]$ is the expected execution time of application t on core m .

In terms of resource allocation problem, the goal of this work is to assign the applications on P , the set of mM processor packages so that the total completion time (i.e. makespan) of the last executed application over all the machines is minimum and the efficient use of energy is maximized. The completion time of a core m is defined as the time in which core m will finalize the processing of the previous assigned application as well as of those already planned applications for the processing elements. This parameter measures the previous workload of a core. Notice that this definition requires knowing both the ready time for a core and the expected time to complete of the tasks assigned to the machine.

III. THE PROPOSED APPROACH

The proposed solution is based on the Min-Min resource allocation algorithm [2], [3]. It is a two phase greedy heuristic. The algorithm starts with a set of all unmapped applications and iteratively assigns tasks to processing elements by computing their expected minimum completion time. For each application this is done by first tentatively scheduling it to each core and estimating the application's completion time on each core. Also for each task, a metric function f_1 (i.e., the core that has the minimum completion time for each application is selected) is computed over all expected completion times. Then the application/core pair

with the best metric match is selected by using a selection function f_2 (i.e., the application with the overall minimum completion time is selected) After that, the application is mapped to the core. Again, this process is repeated with the remaining unmapped applications.

We modified the Min-Min algorithm and adapt it to the contention problem. This modification lies essentially with the calculation of the quality of an mapping of an application to a core machine. The new scoring function takes into account potential memory contention and the voltage-frequency scaling of the individual cores and processors. In addition to all the task-to-core mappings considered, we consider the different voltage-frequency operating points. The source code to the algorithm is freely available [10].

The score of each mapping is calculated with

$$score = \alpha \times MCT + (1 - \alpha) \times Energy, \quad (1)$$

where MCT is the time the core takes to execute all its currently assigned tasks, and α sets the tradeoff between the objectives: energy and MCT . The energy spent executing the this task is calculated with the relation

$$Energy = V^2 \times CT, \quad (2)$$

where CT is the time needed to execute that task on this core. The total completion time is the sum of the completion time of each task, on this core is

$$MCT(m) = \sum_{i=1}^n CT_i(m). \quad (3)$$

As an hypothesis, the time needed to execute each task t on each core c , $CT_i(c)$, is known. These times are based on a dedicated machine, at maximum processor speed. They are presented in an estimated time to compute (ETC) matrix. The machines in our model are heterogeneous. The model is also inconsistent: one core may be faster than another to compute one task, yet slower than another core for another task.

The memory contention and voltage-frequency scaling are introduced as modifying factors to the ETC. Both of these can degrade the completion time of the task, as described in the following subsections.

A. Memory contention penalty

The shared resource we model is the access to main memory. We define a memory contention when several memory-bound tasks are concurrently executed on the same machine. We suppose that the memory requests are not met with the various cache memories. Min-min maps tasks to cores or processors, without specifying any execution order. This prevents the identification of a concurrent execution of memory-bounded tasks. To re-conciliate these two views, we use a statistical approach. When considering the mapping of a memory-bound task to a core, we calculate the percentage of the time the other cores on the same machine spend

Parameter	Value
Tasks	36
Cores	6
Distinct machines	4
% of memory-bound tasks	50

Table I
PARAMETERS

executing memory-bound tasks. This reflects the probability for this task to execute concurrently with another.

$$Penalty_M = 1 + \sum_{c=1}^n pct_c. \quad (4)$$

The original *ETC* time is then multiplied by $Penalty_M$, to reflect the impact of memory contention. For example: on a 2 core single processor, if one core is executing only memory bound tasks, then mapping another memory bound task on the other core results in doubling the basic *ETC* for this task on this core. As mentioned earlier, this is in line with actual measurements made with a memory intensive benchmark application.

B. Voltage-frequency scaling penalty

We assume that each core can operate at a distinct voltage-frequency. The frequency directly influences the performance of the core (lower frequency means greater *CT*), and the voltage its energy consumption. We suppose that each core of a processor can operate at different voltage-frequencies, and that memory-bound tasks are not affected by frequency changes; their *CT* do not change. Examining this latter hypothesis is part of the future work.

IV. EXPERIMENTATION

In this section we present the setup for our experiment, and the results obtained.

A. Parameters

The parameters are summarized in I. There are four machines, two of them are composed of a single core processor, and the other machines of two cores or SMP processors. About half of the tasks are memory-bound.

We provide the corresponding *ETC* matrix [10], for the basic execution times of the tasks.

The model for our voltage-frequency scaling is presented in table II. These voltage-frequency operating points are arbitrarily chosen.

B. Results

In this section we present the results obtained with our proposed algorithm, and a point of comparison with another heuristic from the literature.

Figures 1 and 2 show the effect of the parameter α on the makespan and energy consumption. We can observe that

Operating point	Performance penalty	Voltage (V)
0	$\times 2.0$	0.5
1	$\times 1.5$	1.0
2	$\times 1.0$	2.0

Table II
VOLTAGE-FREQUENCY OPERATING POINTS.

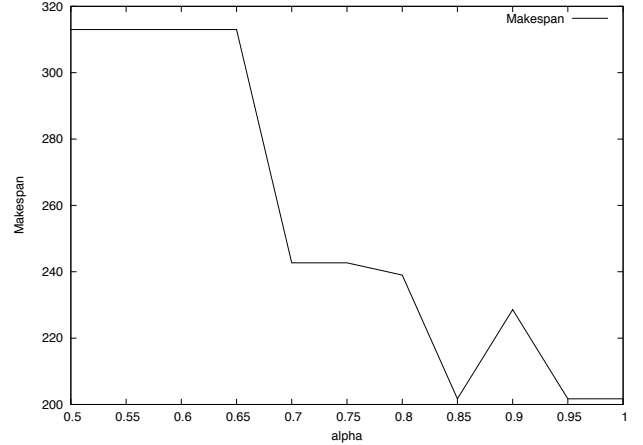


Figure 1. Makespan optimization with memory contention.

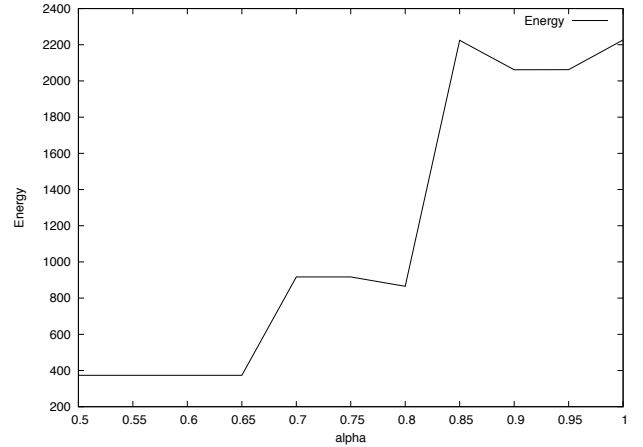


Figure 2. Energy optimization with memory contention.

low α values privilege low energy schedules, to the detriment of makespan, while high α values privilege good makespan results, at the cost of increased energy consumption.

The interesting result is when comparing these results to the original Min-min. This heuristic ignores voltage-frequency scaling and memory contention, so the purpose of this comparison is to measure the impact of our changes on the original heuristic.

To compare the algorithms, we execute the original Min-min on the same *ETC* instance. We then correct the Min-min computed makespan by taking into account the memory contention effect, as described in III-A. This increases

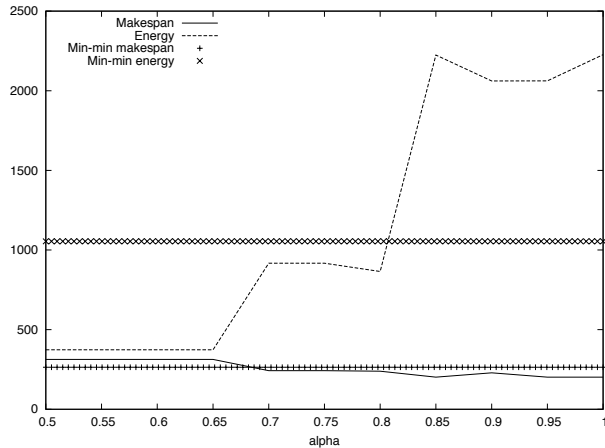


Figure 3. Energy optimization with memory contention.

the makespan value from 195.4 to 263.64 units of time. Similarly, we also compute the total energy consumption from 2. We choose the maximum voltage-frequency point of operation, and the corrected makespan CT . This results in a total energy consumption of 1054.56 J.

For the best trade-off parameter α of 0.8 our approach improves the results for both of the objectives regarding the classical Min-min. We improve the makespan by 9% and the energy consumption is improved by 17%.

V. CONCLUSION

In this study we extended a well-known scheduling heuristic to include energy efficiency and memory contention. The result is a double-objective heuristic which successfully minimizes energy consumption and makespan. Future works include a study on how frequency can impact the performance of memory bound tasks (it is often reported as negligible), and the further validation of the proposed model. This validation will compare the computed makespan and energy consumption from our model with measured values from real experiments.

REFERENCES

- [1] A. Merkel and J. Daly and F. Bellosa, Resource-conscious Scheduling for Energy Efficiency on Multicore Processors, Fifth ACM SIGOPS EuroSys Conference, 2010.
- [2] O. H. Ibarra and C. E. Kim, Heuristic Algorithms for Scheduling Independent Tasks on Non-identical Processors Journal of the ACM, Vol. 24, No. 2, pp. 280–289, 1977
- [3] T. D. Braun and H. J. Siegel and N. Beck and L. L. Bölöni and M. Maheswaran and A. I. Reuther and J. P. Robertson and M. D. Theys and B. Yao and D. Hensgen and R. F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput., Vol. 61, No. 6, pp. 810–837, 2001

- [4] A. Fedorova and S. Blagodurov and S. Zhuravlev, Managing contention for shared resources on multicore processors, Commun. ACM, Vol. 53, No. 2, pp. 49–57, 2010
- [5] A. Ghafoor and J. Yang, Distributed Heterogeneous Supercomputing Management System IEEE Comput., Vol. 26, No. 6, pp 78–86, 1993
- [6] M. Kafil and I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems”, IEEE Concurrency, Vol. 6, No. 3, pp 42–45, 1998
- [7] S. U. Khan and I. Ahmad, A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids IEEE Trans on Parallel and Distributed Systems, Vol. 21, No. 4, pp. 537–553, 2009
- [8] A. Snaveley, D.M. Tullsen, and G. Voelker. Symbiotic jobs cheduling with priorities for a simultaneous multithreading processor, In Int Conf. on the Measurement & Modeling of Computer Systems (SIGMETRICS), pages 6676, 2002.
- [9] STREAM: Sustainable Memory Bandwidth in High Performance Computers, <http://www.cs.virginia.edu/stream/>
- [10] <http://greenit.gforge.uni.lu/mags.html>