Metaheuristics for multiobjective energy-aware scheduling in heterogeneous computing systems

Santiago Iturriaga, Sergio Nesmachnow, Carlos Tutté
Universidad de la República, Uruguay
Email: {siturria,sergion,ctutte}@fing.edu.uy

1 Introduction

The allocation of tasks to be executed in distributed heterogeneous computing (HC) infrastructures is a key problem to solve in order to take full advantage of the computing power of the distributed resources. Nowadays, energy efficiency has become a great challenge in distributed high performance computing, and researchers have focused on developing energy-aware scheduling algorithms for distributed HC infrastructures.

This article reports the advances on applying metaheuristic algorithms to solve the scheduling problem that proposes the simultaneus optimization of makespan and energy consumption in HC systems. The proposed methods include Multithreading Local Search (MLS), a highly efficient multiobjective local search; and two well-known multiobjective evolutionary algorithms (MOEAs), namely NSGA-II and SPEA2. The three methods follow a fully multiobjective approach, since they do not optimize an aggregated function of the problem objectives, but they use Pareto-based dominance techniques in the optimization.

The multiobjective metaheuristics are compared to well-known deterministic heuristic methods over a large set of instances. The experimental results show that the three methods are able to compute accurate schedules in short execution times.

2 Problem formulation

Given a HC system composed of a set of heterogeneous machines $P = \{m_1, \ldots, m_M\}$; each machine with a a certain number of cores m_c, processing speed, and energy consumption; and a collection of tasks $T = \{t_1, \ldots, t_N\}$ to be executed on the system. Let there be an *execution time function* $ET : T \times P \rightarrow \mathbf{R}^+$, where $ET(t_i, m_j)$ is the time required to execute task t_i on one core of the machine m_j; and an *energy consumption function* $EC : T \times P \rightarrow \mathbf{R}^+$, where $EC(t_i, m_j)$ is the energy required to execute task t_i on one core of the machine m_j, and $EC_{IDLE}(m_j)$ is the energy that machine m_j consumes in idle state.

The ME-HCSP proposes to find a schedule $f : T^N \rightarrow P^M$ that simultaneously minimizes the *makespan*, i.e. the total time to execute a bunch of tasks (Eq. 1), and the *energy consumption* (Eq. 2). The energy for executing a given task depends on the execution time, but these two objectives are usually in conflict, since fast machines generally consume more energy than the slower ones.

$$\max_{m_j \in P} \sum_{\substack{t_i \in T: \\ (t_i, m_j) \in f}} ET(t_i, m_j) \quad (1)$$

$$\sum_{\substack{t_i \in T: \\ (t_i, m_j) \in f}} EC(t_i, m_j) + \sum_{m_j \in P} EC_{IDLE}(m_j) \quad (2)$$

Two versions of the ME-HCSP have been tackled in this work, by considering single-core and multi-core machines respectively. In the multi-core version, a single machine can execute more than one task at the same time, provided the number of cores available.

3 Proposed metaheuristics

This section details the proposed metaheuristics to efficiently solve the ME-HCSP. MLS was proposed for the single-core version of the problem, while the MOEAs were applied to solve the multi-core version.

3.1 Multithreading Local Search

MLS is a population-based local search algorithm for the single-core ME-HCSP that maintains a population of non-dominated schedules in order to avoid biasing the search toward one of the objectives of the problem.

The algorithm uses a pool of threads in which each thread is a peer, and no thread performs a master role. Each thread starts by initializing a schedule in the population using a randomized version of the Minimum Completion Time heuristic. After that, each thread loops over the schedules in the population applying a LS to improve them. The local search performed by each thread is based on a randomized version of the Problem Aware Local Search method [1].

MLS is implemented in GNU C++ 4.6. The multithreading support is provided by the GNU POSIX thread library 2.13 and a thread-safe Mersenne Twister method is used for the generation of pseudorandom numbers.

3.2 NSGA-II

NSGA-II is a well-known EA by Deb et al. [2]. The proposed NSGA-II for the ME-HCSP solves the multi-core version of the problem and is implemented using jMetal, a Java framework for multiobjective optimization. The population is initialized using randomized versions of well-known list scheduling heuristics.

dimension	512 × 16				1024 × 32				2048 × 64			
objective	makespan		energy		makespan		energy		makespan		energy	
algorithm	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*	*avg.*	*best*
MLS	10.2%	14.9%	5.8%	7.8%	11.1%	17.2%	6.3%	8.7%	10.0%	19.8%	6.8%	9.4%
NSGA-II	15.4%	33.3%	10.0%	22.0%	16.2%	28.1%	10.4%	18.9%	14.5%	38.2%	9.3%	25.0%
SPEA2	15.2%	33.3%	9.4%	21.9%	15.9%	28.1%	9.7%	17.6%	13.9%	36.5%	8.5%	24.0%

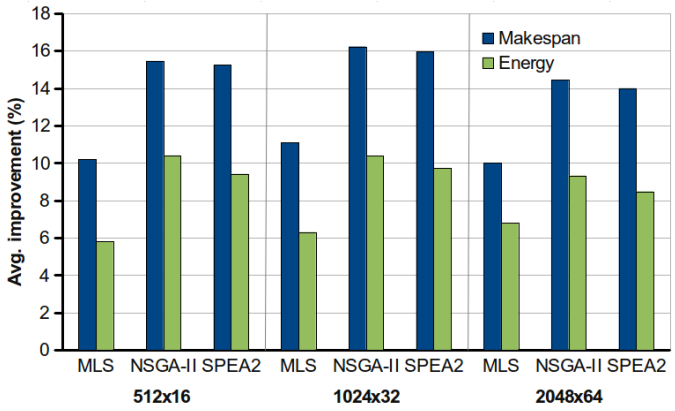

Figure 1: Average improvement over the best deterministic heuristic.

3.3 SPEA2

The Strength Pareto Evolutionary Algorithm 2 (SPEA2) is also a well-known evolutionary algorithm proposed by Zitzler et al. [4]. Like NSGA-II, the proposed SPEA2 for the ME-HCSP solves the multi-core version of the problem and is implemented using jMetal. The initial population is initialized exactly the same as the NSGA-II.

4 Experimental analysis

The experimental analysis of the proposed methods compares the makespan and energy results with the Sufferage heuristic and four multiobjective variants of the Min-Min heuristic [3]. All the experiments were performed in a 24-core machine with AMD Opteron 6172 Processors at 2.1GHz and 24 GB of RAM.

A total number of 792 instances were used in the MLS experimental analysis, and 55 instances where used to evaluate the proposed MOEAs. The parameter configuration study performed on 4 small instances showed that the best configuration for MLS consists in using 24 concurrent threads, a population of 30 individuals, and a stopping criterion of 10 seconds. On the other hand, the proposed MOEAs computed the best results when using a population of 150 individuals for NSGA-II and 100 individuals for SPEA2, and both using a stopping criterion of 60 seconds.

Table 1 reports the average and best improvements over the best deterministic heuristic for each problem dimension, computed in 30 independent executions performed for each algorithm. Figure 1 summarizes the average improvements by dimension.

For the MLS algorithm, the results in Table 1 indicates that disregarding the problem dimension, an average improvement above of 10% in makespan and 6% in energy can be obtained in only 10 seconds execution time. A scalability analysis showed a promising 19.4 speedup value when using 24 threads. The good scalability behavior of MLS indicate further experiments should be performed increasing the number of threads and increasing the dimension of the problem instances.

The experimental analysis also demonstrate that both proposed MOEAs compute accurate solutions in bounded execution time. Average improvements up to 18.6% in makespan and up to 11.9% in energy where computed by the NSGA-II algorithm, and similar improvements up to 18.3% in makespan and up to 11.5% in energy where computed by the SPEA2 algorithm. Even though the MOEAs require substantially longer execution times, the improvements over the MLS are not overly significant; this results suggest that the multi-core problem is in fact harder to solve than the single-core problem.

Because of the reduced execution time demanded to compute accurate schedules, MLS is a candidate to be considered when hybridizing EA with LS. As future work we propose to compare our current MOEAs results with an improved version of hybrid NSGA-II+MLS and SPEA2+MLS, by including MLS as an internal operator of the proposed MOEAs.

References

[1] E. Alba and G. Luque. A new local search algorithm for the DNA fragment assembly problem. In *Proc. of 7th European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 1–12, 2007.

[2] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, PPSN VI, pages 849–858, London, UK, UK, 2000. Springer-Verlag.

[3] P. Luo, K. Lü, and Z. Shi. A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. *J. Parallel Distrib. Comput.*, 67(6):695–714, 2007.

[4] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems*, pages 95–100, 2002.