Affinity multiprocessor scheduling considering communications and synchronizations using a Multiobjective Iterated Local Search algorithm

S. Nesmachnow[1] **and A. Tchernykh**[2]

[1] Universidad de la República, Uruguay
[2] CICESE, Ensenada, Baja California, México

E-mail: sergion@fing.edu.uy, chernykh@cicese.mx

Abstract. This article studies the affinity scheduling problem in multicore computing systems, considering the minimization of communications and synchronizations. The problem consists in assigning a set of tasks to resources to minimize the overall execution time of the set of tasks and the execution time required to compute the schedule. A Multiobjective Iterated Local Search method is proposed to solve the studied affinity scheduling problem, which considers the different times required for communication and synchronization of tasks executing on different cores of a multicore computer. The experimental evaluation of the proposed scheduling method is performed over realistic instances of the scheduling problem, considering a set of common benchmark applications from the parallel scientific computing field, and a modern multicore platform from National Supercomputing Center, Uruguay. The main results indicate that the proposed multiobjective Iterated Local Search method improves up to **21.6%** over the traditional scheduling techniques (a standard Round Robin and a Greedy scheduler)

1. Introduction

Scheduling is a key problem on Heterogeneous Computing (HC) systems [1]. Modern multicore processors are a special case of HC platforms. Even though they are built using identical processing units with the same computing power, major processor manufacturers adopted the Non-Uniform Memory Architecture (NUMA) design (AMD from 2003 and Intel from 2007). In a NUMA design, the main memory is organized in a set of separate chips, to reduce the bottleneck impact of simultaneous memory accesses. However, the time required for memory access varies for different cores in the system, and different communication and synchronization speeds exist for parallel tasks that cooperate to solve a problem.

Affinity scheduling [2] are specific planning methods for multipocessor systems that propose assigning tasks to cores, taking advantage of the capabilities of tasks to execute faster in certain cores due to data locality, cache utilization, or communications with other tasks. The main goal of an affinity scheduling technique is reducing the negative impact of common overheads that appear in parallel scientific computing, e.g., due to task communication and synchronization, dynamic resource management, or load balancing [3].

Traditional scheduling problems are NP-hard, thus exact resolution techniques are only useful to solve small problem instances (i.e., few tasks, few resources). Metaheuristics [4] are efficient search methods to compute accurate solutions in reasonable execution times. This is relevant when considering that schedulers must operate in real time.

In this line of work, this article proposes a multiobjective metaheuristic approach to solve the assignment problem related to affinity scheduling in multicore NUMA systems. A Multiobjective Iterated Local Search (MILS) scheduler is introduced to improve tasks efficiency by minimizing communication/synchronization times and the effective time required to compute the schedule. The experimental evaluation of the proposed method is presented for real parallel scientific computing benchmark applications, considering different topologies and communication patterns, and also real multicore computing platforms. Results indicate that the proposed multiobjective Iterated Local Search method improves up to **21.6%** over traditional scheduling techniques used as baseline for the comparison.

2. The multiobjective affinity scheduling problem in multicore systems

The mathematical model of the affinity scheduling problem to minimize communications and synchronizations in multicore computing systems (ASP-CS) considers the following elements:

- A multicore system with a set of cores $N = \{n_1, \ldots, n_a\}$.
- A set of tasks $T = \{t_1, \ldots, t_b\}$ to be executed on the system.
- A *communication function* $C : T \times T \rightarrow \mathbb{N}^+$, where $C(t_i, t_j)$ indicates the number of communications between two tasks t_i and t_j, $1 \leqslant i \leqslant b$, $1 \leqslant j \leqslant b$.
- A *synchronization function* $S : T \times T \rightarrow \mathbb{N}^+$, where $S(t_i, t_j)$ indicates the number of synchronizations between two tasks t_i and t_j, $1 \leqslant i \leqslant b$, $1 \leqslant j \leqslant b$.
- A *communication cost function* $CC : N \times N \rightarrow \mathbb{R}^+$, where $CC(n_h, n_k)$ is the time required to communicate tasks executing in cores n_h and n_k, $1 \leqslant h \leqslant a$, $1 \leqslant k \leqslant a$.
- A *synchronization cost function* $SC : N \times N \rightarrow \mathbb{R}^+$, where $SC(n_h, n_k)$ is the time required to synchronize tasks executing in cores n_h and n_k, $1 \leqslant h \leqslant a$, $1 \leqslant k \leqslant a$.

The ASP-CS problem proposes finding a scheduling function $f : T \rightarrow N$ to assign tasks to cores in the multicore system ($f(t_i) = n_h$ indicates that task t_i is assigned to execute on core n_h, $1 \leqslant i \leqslant b$ and $1 \leqslant h \leqslant a$), to minimize the total time demanded for communication and synchronizations between tasks (Eq. 1) and minimize the overall execution time to compute the schedule, in order to provide an accurate method to be applied in real time. The ASP-CS problem follows a non-preemptive model: each task is considered as an atomic processing unit, which cannot be divided nor interrupted.

$$\sum_{t_i \in T} \sum_{t_j \in T} C(t_i, t_j) \times CC(f(t_i), f(t_j)) + S(t_i, t_j) \times SC(f(t_i), f(t_j)) \tag{1}$$

ASP-CS is a NP-hard combinatorial optimization problem [5] and traditional exact methods are not useful to solve large instances of the problem in reduced execution times. Thus, more efficentoptimization methods, such as metaheuristics, must be applied to solve problen instances as close as possible to real time.

3. The proposed multiobjective Iterated Local Search scheduler

MILS is a metaheuristic method that extends the traditional local search by including a perturbation operator to escape from local optima and a multiobjective evaluation considering Pareto dominance.

Algorithm 1 presents a pseudocode of the proposed scheduler. The search starts from a randomly generated initial solution. MILS is an iterative method; in each iteration, the current candidate solution is perturbed and improved using a local search that involves different tasks movements, proved to be useful for HC scheduling [6]: random move, move to the least loaded resource, move from the most loaded resource and swap. The perturbation is a diversification operator, whose main goal is to provide MILS the capability of escaping from local optima. The acceptance criterion applies Pareto dominance to determine if the current solution is replaced by the best solution found in the neighborhood of the perturbed solution. The iterative process repeats until a given stopping criteria (e.g., a predefined effort stopping criterion or a stagnation detection) is met.

Algorithm 1 Schema of the proposed MILS sceduler.

1: $s_0 \leftarrow$ GenerateInitialSolution()
2: $s^* \leftarrow$ LocalSearch(s_0)
3: **repeat**
4: $s' \leftarrow$ Perturbation(s^*)
5: $s'' \leftarrow$ LocalSearch(s')
6: $s^* \leftarrow$ ParetoBased-AcceptanceCriterion(s^*,s'')
7: **until** stop criterion is met
8: **return** s^*

The proposed MILS aims at providing an efficient real-time scheduler for modern multiprocessors, which dynamically incorporates information about the processor architecture and also topological information about the application, provided by the user. The main features of the processor architecture (number of cores, topology, and organization) are obtained at runtime using the `hwloc` tool [7]. `hwloc` provides qualitative and quantitative information about the computing elements and the underlying architecture via an Application Programming Interface developed in the C language. Using runtime information allows the proposed scheduler to be integrated in modern Resource Management Systems for cluster and other High Performance Computing infrastructures.

4. Experimental evaluation

The evaluation of the proposed scheduler was performed considering problem instances of three real scientific computing applications with different communication patterns:

- *Heat*: A numerical analysis application describing the evolution of temperature in a bar. The application solves the differential equation governing the heat transfer process applying a master/slave parallel model and domain decomposition. The communication pattern follows a flat topology, where a master process sends data to a set of slaves processes for computation; slaves compute and send back the results to the master, which reduces the results. Slaves communicate with each other to deal with computations in the borders of the domain decomposition. The topology is described in Figure 1, where M is the master, S_i are the slave processes, and communications costs are the labels of each edge.
- *Workflow*: Describes a generic workflow application, where data are distributed and processed by several independent tasks, dependencies between them, according to a specific flow. No distinguished process controls the flow, but critical path determines the efficiency of the application. Communications and synchronizations are defined according a level-based synchronization pattern, as described in Figure 2. Blue rectangles represents synchronizations and the corresponding synchronization costs are marked in blue.

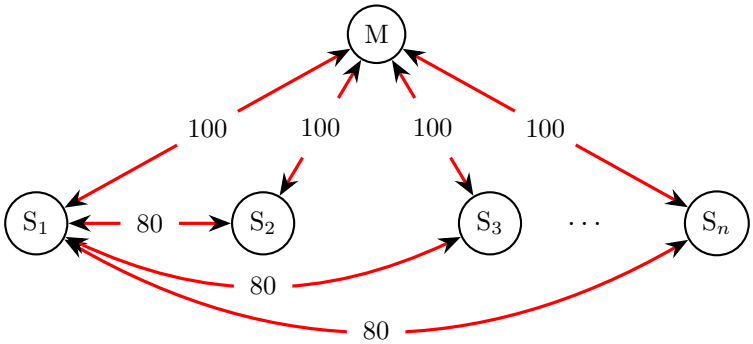

Figure 1. Flat topology.

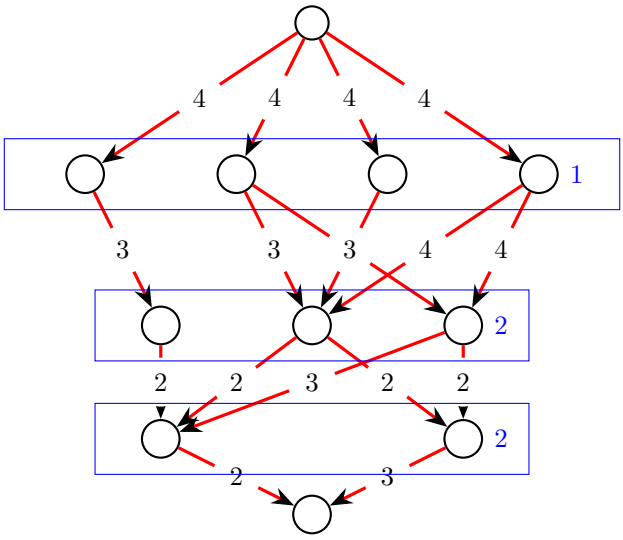

Figure 2. Application-driven topology with level-based synchronizations.

- *Quicksort*: This sorting algorithm follows a topology with a hierarchical order defined between tasks. Each task communicates exclusively with tasks in the nearest levels in the hierarchy. Synchronizations are performed between tasks in the same level. The topology is that of a tree, where communications are performed between parent and children and synchronizations are between siblings, as described in Figure 3.

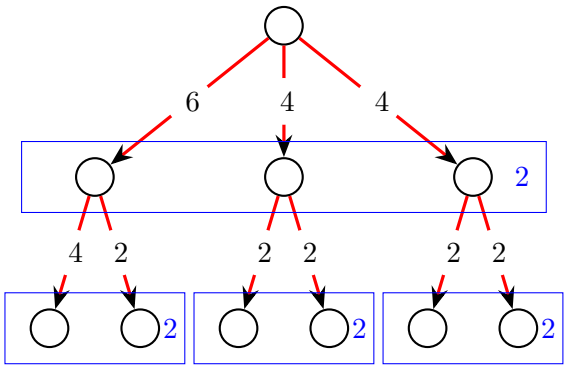

Figure 3. Hierarchical topology (tree) with synchronizations between siblings.

Experiments were performed over a Intel Xeon Gold 6138 multicore server from National Supercomputing Center (Cluster-UY), Uruguay [8]. Cluster-UY is a relevant high performance computing platform in the Latin-American ecosystem for e-science, executing many large-scale scientific applications that can take advantage of accurate scheduling to compute better results in reduced execution times [9]. The hardware architecture of the considered server is presented in Figure 4, as reported by the `hwloc` tool.

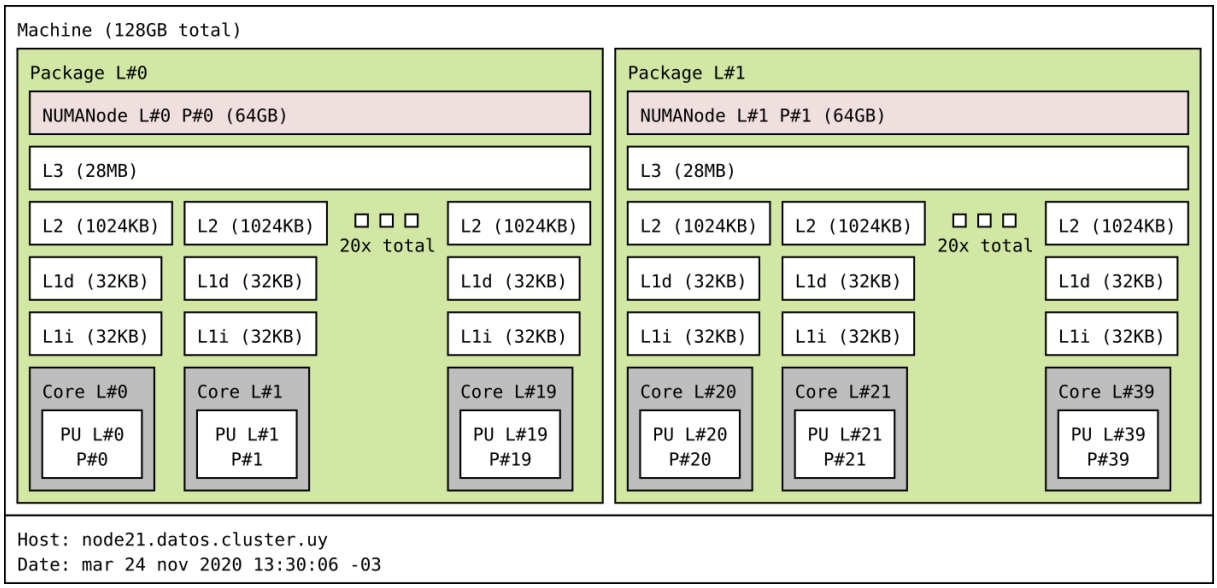

Figure 4. Hardware specification of the considered multicore computer from Cluster-UY, Uruguay, obtained using the `hwloc` tool.

Table 1 reports the main results of the experimental evaluation of the proposed scheduler. Results correspond to 30 independent executions of MILS, using different number of tasks (b) and cores (a). Results of the proposed MILS are compared with a a greedy heuristic (GH) and a traditional Round Robin (RR) method [10]. Improvements over the baseline solutions, regarding the total execution time of the set of tasks considered and reported. The (percentage) improvement over baseline heuristic H is defined as $\Delta H = (f_H - f_{MILS})/f_{MILS}$. The best improvements found are marked in bold font.

Table 1. MILS improvements over GH and RR baseline schedulers.

$b \times a$	*Heat transfer*		*Workflow*		*Quicksort*	
	ΔGH	ΔRR	ΔGH	ΔRR	ΔGH	ΔRR
64×6	12.8%	**21.6%**	9.3%	**13.2%**	7.7%	**16.4%**
64×12	8.0%	**18.4%**	2.0%	**9.7%**	3.9%	**5.9%**
64×24	3.6%	**8.4%**	8.5%	**10.0%**	3.2%	**5.4%**
128×8	10.1%	**19.5%**	1.5%	**7.8%**	3.9%	**7.0%**
128×16	6.7%	**12.4%**	5.3%	**10.5%**	4.3%	**8.3%**
128×32	2.3%	**5.9%**	8.2%	8.9%	3.0%	**6.6%**
256×16	10.8%	**18.9%**	9.4%	**16.1%**	7.3%	**11.0%**
256×32	7.4%	**15.2%**	8.3%	**13.9%**	4.3%	**10.5%**
256×64	3.2%	**6.1%**	6.7%	13.2%	3.7%	**7.4%**

Results on Table 1 demonstrate that the proposed MILS is able to compute accurate schedules regarding both execution time of the batch of tasks and the scheduler execution time. Improvements over the traditional RR heuristic were up to 21.6% when considering 64 tasks to be scheduled in a six-cores server. Improvements over GH were up to 12.8%. The best improvements were obtained for problem instances with the highest ratio of tasks per available core. Solutions computed by MILS clearly dominated GH, as reported in the dominance analysis presented in Figure 5 for a representative case study. MILS is able to better capture the main features of the underlying architecture and properly explore the search space of the problem.

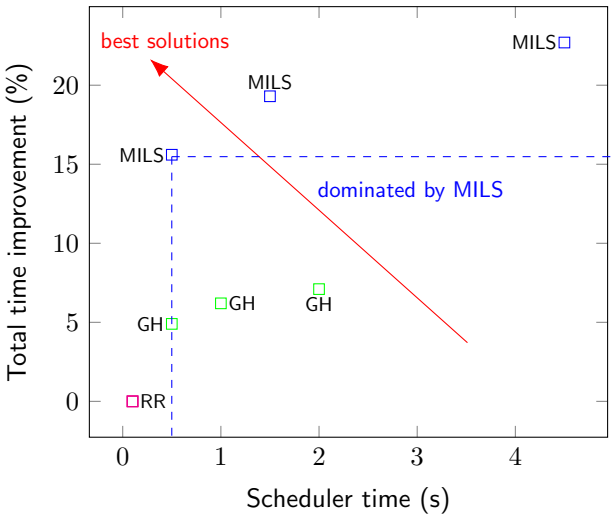

Figure 5. Dominance analysis for a representative problem instance.

5. Conclusions and future work

This article addressed the affinity scheduling problem of parallel applications in multicore systems to minimize the execution time of a batch of tasks, considering communications and synchronizations between processes, and the execution time required to compute the schedule.

A MILS metaheuristic was proposed to solve the scheduling problem, including specific Pareto-based local search operators and considering Pareto dominance as acceptance criterion for solutions found in the search. The experimental evaluation of the proposed MILS was performed over realistic instances of the scheduling problem, considering parallel scientific applications with different communication/synchronization patterns and a modern multicore platform from National Supercomputing Center, Uruguay. The main results of the evaluation indicate that the proposed MILS is an effective method for affinity scheduling, improving up to 21.6% over traditional scheduling techniques.

The main lines for future work are related to extend the evaluation of the proposed multiobjective scheduler, by considering other realistic applications and multicore architectures, and the use of automatic profiling tool to characterize the considered applications.

References

[1] Nesmachnow S, Cancela H and Alba E 2010 *Soft Computing* **15** 685–701
[2] Markatos E and LeBlanc T 1994 *IEEE Transactions on Parallel and Distributed Systems* **5** 379–400
[3] Subramaniam S and Eager D 1994 Affinity scheduling of unbalanced workloads *ACM/IEEE Conference on Supercomputing* pp 214–226
[4] Nesmachnow S 2014 *International Journal of Metaheuristics* **3** 320
[5] Ullman J 1975 *Journal of Computer and System Sciences* **10** 384–393

[6] Nesmachnow S, Cancela H and Alba E 2012 *Applied Soft Computing* **12** 626–639
[7] Goglin B 2014 Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc) *High Performance Computing & Simulation* pp 74–81
[8] Nesmachnow S and Iturriaga S 2019 Cluster-UY: Collaborative Scientific High Performance Computing in Uruguay *Supercomputing* Communications in Computer and Information Science (Springer International Publishing) pp 188–202
[9] Gitler I, Gomes A and Nesmachnow S 2020 *Communications of the ACM* **63** 66–71
[10] Regueira D, Iturriaga S and Nesmachnow S 2017 Communication-aware affinity scheduling heuristics in multicore systems *High Performance Computing* Communications in Computer and Information Science (Springer International Publishing) pp 33–48