Heuristic-based Task-to-Thread Mapping in Multi-Core Processors

Mohammad Samadi Gharajeh[1,*], Sara Royuela[2], Luis Miguel Pinho[1], Tiago Carvalho[1], Eduardo Quiñones[2]
[1]School of Engineering, Polytechnic Institute of Porto, Portugal
{mmasa, lmp, tdc}@isep.ipp.pt
[2]Barcelona Supercomputing Center, Barcelona, Spain
{sara.royuela, eduardo.quinones}@bsc.es

***Abstract*— OpenMP can be used in real-time applications to enhance system performance. However, predictability of OpenMP applications is still a challenge. This paper investigates heuristics for the mapping of OpenMP task graphs in underlying threads, for the development of time-predictable OpenMP programs. These approaches are based on a global scheduling queue, as well as per-thread allocation queues. The proposed method is divided into scheduling and allocation phases. In the former phase, OpenMP task-parts are discovered from OpenMP graph and placed in the scheduling queue. Afterwards, an appropriate allocation queue is selected for each task-part using four heuristic algorithms. In the latter phase, the best task-part is selected from the allocation queue to be allocated to and executed by an idle thread. Preliminary simulation results show that the new method overcomes BFS and WFS in terms of scheduling time and idle time.**

Keywords—real-time systems, OpenMP, task-to-thread mapping, scheduling time, idle time

I. INTRODUCTION

Multi-core processors can be used in real-time embedded systems to optimize the efficiency of modern applications, such as advanced driver-assistance systems (ADASs). In these systems, the computer-based controller must respond within a bounded time. So, in real-time computing, the correctness of the system depends not only on the computation throughput, but also on the response time in which the result is produced. However, as multi-cores behave like a black-box, time-predictability of the system is a big challenge [1].

OpenMP is an industry de-facto standard to parallelize C, C++, and Fortran programs widely used in high-performance computing (HPC). Early versions of the model targeted data-parallelism in loop intensive applications for shared-memory architectures by describing a *thread-centric model* exposing details about the scheduling behavior of the application. That prescriptive nature has evolved into a more descriptive one in latter extensions, including a powerful *tasking model* that allows describing complex types of fine-grained and irregular parallelism [3] and a device model, built on top of it, for offloading to GPUs and other accelerators [2]. Furthermore, OpenMP is increasingly being adopted in embedded parallel and heterogeneous systems as well.

OpenMP tasks are units of scheduling that can be further divided in task-parts, when task include *task scheduling points* (TSP). TSPs are points at which the runtime system can suspend a task in favor of another one, to later resume it. In OpenMP, tasks can be defined as *tied* or *untied*. All task-parts of a tied task must execute on the same thread, while task-parts of an untied task can be executed on different threads. The current tasking model enables the programmer to define explicit tasks and the data dependencies existing between them. Tasks are executed by a team of threads, one as a master thread and others as worker threads, which allows the system to efficiently utilize many-core architectures while hiding their complexity from the programmer. Basically, the available implementations rely on breadth-first (BFS) and work-first (WFS) schedulers. However, the OpenMP specification allows the implementation of any task-to-thread scheduler [3] that meets the specification.

A few existing schedulers for task-to-thread mapping [4] use heuristic techniques. The main objective of this paper is to present efficient schedulers using heuristic algorithms based on a multi-queue structure through the scheduling and allocation phases. In the scheduling phase, task-parts are discovered from the OpenMP-DAG that represents the application and placed into a global scheduling queue shared among all threads. Note that each thread includes an allocation queue. Then, an appropriate allocation queue is selected for each task-part and placed in the scheduling queue using the suggested heuristic scheduling algorithms. In the allocation phase, when a thread is idle and its allocation queue contains any task-parts, one of them is selected based on the suggested heuristic allocation algorithms.

II. LITERATURE REVIEW

Serrano et al. [3] showed that timing analysis with tied tasks is worse than that with untied tasks. They considered two scenarios: (i) a generic scheduler (GenS) without considering any concrete scheduling policy, and (ii) the breadth-first scheduler (BFS) and work-first scheduler (WFS) algorithms with first-in, first-out (FIFO) policies. The strategies determine the number of tasks preventing another task from execution using creation time of new tied tasks and resumption time of existing tasks.

Sun et al. [5] designed a technique to schedule synchronous OpenMP tasks. This work divides tasks into several groups and then assigns some of them to dedicated cores, isolating tied tasks. The scheduling process is conducted through the three phases: task grouping, offline task management, and online task management.

Royuela et al. [6] designed a run-time scheduler that uses dynamic information of OpenMP threads and tasks running within several concurrent teams (i.e., concurrent parallel regions). This information may contain the priority list of tasks ready to execute and the list of OpenMP threads waiting in a barrier to cooperatively execute multiple parallel regions.

Wang et al. [7] presented a partitioning-based algorithm in which the tied constraint can be automatically guaranteed while partitioning the OpenMP system to a multiprocessor

This work has been co-funded by the European commission through the AMPERE (H2020 grant agreement N° 745601) project.

* PhD Candidate, Universitat Politècnica de Catalunya, Barcelona, Spain

 DOI 10.1109/ETFA52439.2022.9921453

system. After vertices are partitioned to one processor, they are not allowed to migrate among processors. This work applies an on-line scheduler to schedule subtasks based on real execution time instead of worst-case execution time (WCET).

Melani et al. [4] presented two OpenMP-compliant static allocation techniques satisfying the time-predictability requirements of safety-critical systems. The former technique uses a non-trivial integer linear programming (ILP) formulation to calculate the minimum possible response time obtainable for a given OpenMP-DAG, which provides a computationally expensive but optimal allocation strategy. The latter one provides response times within a much smaller computational complexity compared to the optimal one using well-known sub-optimal heuristic strategies.

The main problems with most of the existing mapping algorithms are that (i) they do not schedule tasks on threads based on global information of the system, and (ii) they do not use temporal conditions of the system (e.g., execution time and deadline) in each phase of the mapping process through heuristic techniques.

III. THE PROPOSED MAPPING METHOD

We propose a mapping method that uses temporal conditions of the system in each phase of the mapping process to meet timing requirements. This method is a fully online mapping technique, so it schedules task-parts whenever they are created in the system. The main objective is to achieve a work-conserving and load-balancing mapping using heuristic algorithms based on temporal information.

A. System model

An OpenMP task system $\mathcal{T}$ is assumed that consists of n tasks $\{\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_{n-1}\}$. $\mathcal{T}$ is characterized by a deadline D that can be determined by the system designer. Each task can be either tied or untied. A task $\mathcal{T}_i$ includes a set of sequentially ordered task-parts $\{P_{i,0}, P_{i,1}, \dots\}$. It can be represented as a *direct acyclic graph* (DAG) $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. A vertex $\mathcal{V}_{i,x}$ in $\mathcal{V}$ belongs to the x-th task-part $P_{i,x}$ of task $\mathcal{T}_i$ and contains an execution time $C_{i,x}$. It is also characterized by a response time $RT_{i,x}$ that can be estimated based on $C_{i,x}$, the sum of the execution time of the task-parts, and the deadline of $\mathcal{T}$. It is assumed that the number of threads is equal to the number of cores.

This method is presented for the system models that do not include any conditional branch. It is assumed that execution time of the tasks is known for the scheduler. Task-parts are executed by threads as non-preemptive units of scheduling, so when a task-part is executed, it is not suspended until completion. The number of OpenMP threads is considered equal to the number of system threads an assigned in a 1-to-1 mapping fashion. All task-parts of a tied task must be executed by one thread, but each task-part of an untied task can be executed by different threads.

There is a global queue for all tasks, called scheduling queue (SQ), to store the task-parts discovered from $\mathcal{T}$. The length of SQ is identified by $SQ_{\mathcal{L}}$ that equals the number of all task-parts as

$$SQ_{\mathcal{L}} = \sum_{i=0}^{n-1} N_i \quad (1)$$

, where N_i is the number of task-parts of $\mathcal{T}_i$. The system includes *m* threads as $\{\mathcal{TH}_0, \mathcal{TH}_1, \dots, \mathcal{TH}_{m-1}\}$, in which each thread has an allocation queue ($\mathcal{AQ}$). Note that both SQ and $\mathcal{AQ}$ include ready task-parts which have not been allocated to threads yet.

Fig. 1 shows an OpenMP program and its DAG. The program consists of 1 implicit task corresponding to the single directive, and 6 explicit tasks. Each task contains one or multiple task-parts. $\mathcal{T}_1$, $\mathcal{T}_4$, and $\mathcal{T}_5$ are child tasks of $\mathcal{T}_0$, $\mathcal{T}_2$ is the child task of $\mathcal{T}_1$, $\mathcal{T}_3$ is the child task of $\mathcal{T}_2$, and $\mathcal{T}_6$ is the child task of $\mathcal{T}_5$. Furthermore, there are two synchronizations: (1) the taskwait in line 19 forces the task-part $P_{1,2}$ of $\mathcal{T}_1$ to be executed after completing the $\mathcal{T}_2$, and (2) the data dependencies in lines 23 and 28 force $\mathcal{T}_5$ to be executed after $\mathcal{T}_4$.

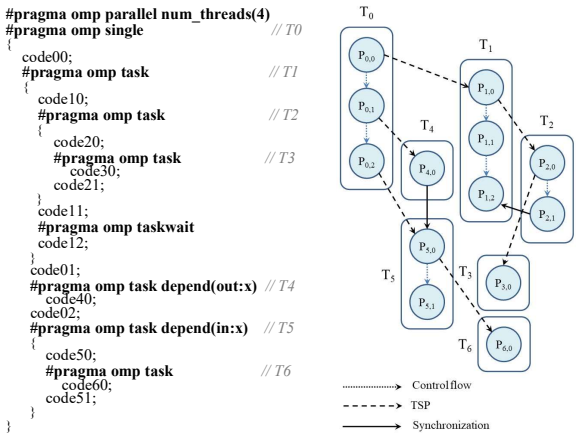


Fig. 1. An OpenMP-DAG example. (a) source code; (b) DAG.

B. The mapping algorithm

Fig. 2 shows the main elements of the proposed mapping algorithm that includes two phases: (1) scheduling and (2) allocation. These phases are carried out simultaneously.

In the scheduling phase, each task-part $P_{i,x}$ is discovered from $\mathcal{T}$ and stored in SQ by all threads, starting at the first vertex $\mathcal{V}_{0,0}$ towards leaves. Afterwards, each task-part is selected from SQ and stored in one of the $\mathcal{AQ}s$ by the master thread. After a task-part is executed by an idle thread, its child tasks, if any, will be appended to SQ. This process is repeated until all task-parts of $\mathcal{T}$ are executed.

A suitable thread (i.e., its corresponding $\mathcal{AQ}$) is selected using one of the following suggested scheduling algorithms:

- *First-fit scheduling heuristic*: the thread (i.e., $\mathcal{AQ}$) containing the minimum number of task-parts for increasing the work-balancing of the queues;
- *Best-fit scheduling heuristic*: the $\mathcal{AQ}$ with the minimum execution time for decreasing the number of missed deadlines;
- *Optimum-fit scheduling heuristic*: the $\mathcal{AQ}$ including the maximum response time (of the task-parts) for increasing the data locality between tasks and reducing the balance of the work on the $\mathcal{AQ}s$;
- *Multi-criteria scheduling heuristic*: the $\mathcal{AQ}$ having the characteristics of the previous heuristics using a multi-criteria decision.

The allocation phase is carried out for each $\mathcal{AQ}$ separately. Accordingly, an appropriate task-part is selected from $\mathcal{AQ}_i$ to be allocated to and executed by the thread TH_i, the remaining task-parts in the queue are rearranged to remove the blank space, and the system info (e.g., the number of task-parts in $\mathcal{AQs}$) is updated.

If $\mathcal{AQ}_i$ includes only one task-part, it will be allocated to and scheduled by the relevant thread directly. Otherwise, the best task-part is selected from the queue using one of the following suggested allocation algorithms:

- *Best-fit allocation heuristic*: the task-part having the minimum execution time for decreasing the number of missed deadlines;
- *Optimum-fit allocation heuristic*: the task-part with the maximum response time for reducing the workload on $\mathcal{AQs}$ and increasing the chance of receiving new task-part;
- *Multi-criteria allocation heuristic*: the task-part including the shortest execution time and the longest response time based on the features of the prior heuristics through a multi-criteria decision.

Algorithm 1 describes the mapping process of the proposed method for tied tasks meeting task scheduling constraint (TSC) 1 [1] of the specification [8]. SCHEDULE_ALGORITHM chooses one of the $\mathcal{AQs}$ using the scheduling heuristics, PRE_THREAD selects the thread belongs to the first task-part in the same task, SUSPEND_TASK includes the list of tasks suspended in the thread, ALLOC_ALGORITHM selects one of the task-parts from the thread's queue, and DESCENDANT indicates whether a new task is the descendant of the suspended tasks. In the case of untied tasks, it is not needed to check the operations at line 18 and lines 25-32.

IV. EVALUATION RESULTS

The performance of the proposed mapping methodology, using the algorithms described in Section III.B, is evaluated under different number of tasks when the number of threads is 4 by comparing the results with those of BFS and WFS, in terms of scheduling time (i.e., end-to-end response time[2]) and idle time of the threads based on the mean of results. For each case, a random graph is generated and scheduled over 10 iterations in which the execution time of each task-part is randomly generated at each iteration and only the average results are considered in the results. The deadline D is determined in the simulation process as following

$$D = \rho * \sum_{j=0}^{N_i-1} \mathcal{ET}_j^i \qquad (2)$$

, where N_i indicates the number of task-parts in T_i, $\mathcal{ET}_j^i$ represents the execution time of the task-part P_j in T_i, and ρ indicates a random number that is considered in the range of [0.5, 1].

Fig. 3 shows the comparison results of the methods for tied tasks. In this case, *best-fit, multi-criteria* is selected as the best heuristic pair. The results show that the new method reduces scheduling time and idle time in most cases where the number of tasks is greater than 10. Since WFS schedules task-parts in only one thread, the results obtained by this algorithm are worse than those achieved by the others.

Fig. 4 illustrates the simulation results for untied tasks in the same scenarios. Similar to tied task, *best-fit, multi-criteria* is chosen as the best heuristic pair. WFS works well where the number of tasks is {10, 15}. But at all, the performance of the new method is obvious, especially when the number of tasks is greater than 15.

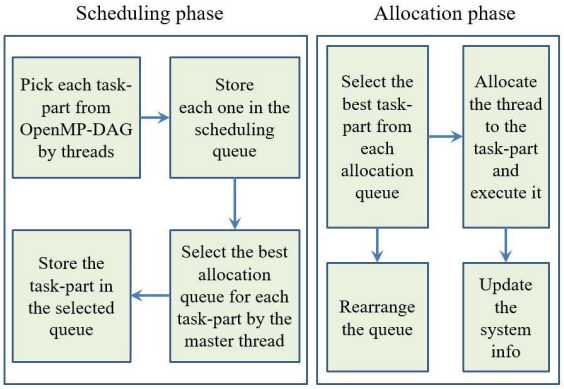
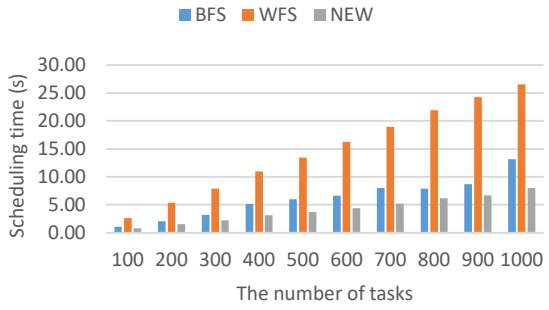


Fig. 2. Main elements of the proposed mapping method.

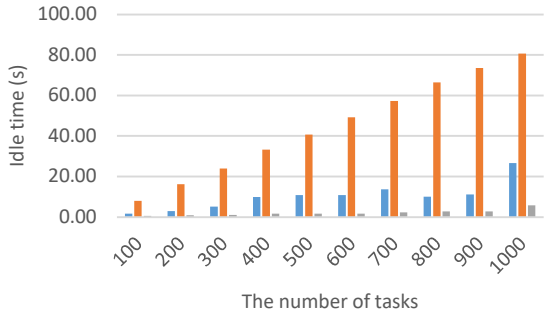
Algorithm 1. Mapping of an OpenMP-DAG application including tied tasks	
1:	$G \leftarrow$ predefined or random graph; $num \leftarrow$ number of parts
2:	$P \leftarrow$ pool of threads; $SQ \leftarrow []$; $AQs \leftarrow []$; $c \leftarrow 0$
3:	**while** $c < num$
4:	**for each** thr **in** P
5:	**if** $thr.status$ == 'busy' **and** $thr.ftime <= t$
6:	$thr.status \leftarrow$ 'i'; $c \leftarrow c + 1$; $c \leftarrow G[c]$
7:	if $p.child$!= NULL
8:	$SQ \leftarrow$ APPEND($p.child$)
9:	if $p.sibling$!= NULL
10:	$SQ \leftarrow$ APPEND($p.sibling$)
11:	**for each** p **in** SQ
12:	**if** $p.dep$ == NULL **or** $p.dep.status$ == 'idle'
13:	**if** $p.pid$ == 0
14:	thr_num $\leftarrow$ IDLE_THREAD(P)
15:	**if** thr_num == NULL
16:	thr_num $\leftarrow$ SCHEDULE_ALGORITHM(P)
17:	**else**
18:	thr_num $\leftarrow$ PRE_THREAD($p.pid$)
19:	$AQs \leftarrow$ APPEND(p, thr_num)
20:	**for each** thr **in** P
21:	**if** $thr.status$ == 'idle'
22:	**if** SIZE($AQs[thr]$) > 0
23:	**if** SUSPEND_TASK(thr).empty()
24:	$p \leftarrow$ ALLOC_ALGORITHM($AQs[thr]$)
25:	**else**
26:	$l \leftarrow []$
27:	**for each** tp **in** SUSPEND_TASK(thr)
28:	$l \leftarrow$ APPEND(tp)
29:	**for each** tp **not in** SUSPEND_TASK(thr)
30:	**if** DESCENDANT(tp, thr) == TRUE
31:	$l \leftarrow$ APPEND(tp)
32:	$p \leftarrow$ ALLOC_ALGORITHM(l)
33:	$thr \leftarrow$ ALLOCATE(p); $thr.status \leftarrow$'s'

[1] TSC1: *Scheduling of new tied tasks is constrained by the set of task regions that are currently tied to the thread and that are not suspended in a barrier region. If this set is empty, any new tied task may be scheduled. Otherwise, a new tied task may be scheduled only if it is a descendent task of every task in the set.*

[2] The difference between the finish time of the last task-part and the start time of the first task-part

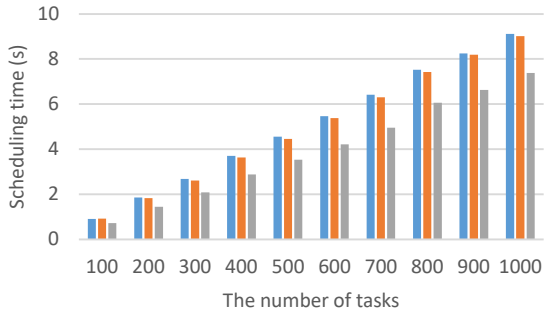


(a)

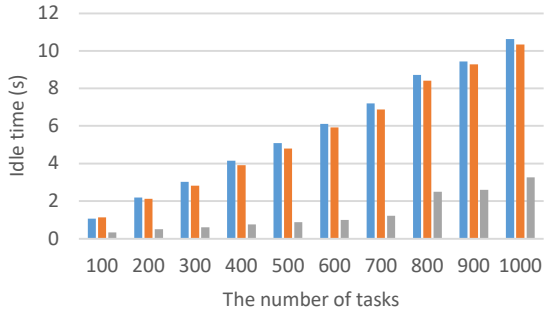


(b)

Fig. 3. Effect of the number of tasks on the performance metrics for tied tasks: (a) scheduling time; (b) idle time.



(a)



(b)

Fig. 4. Effect of the number of tasks on the performance metrics for untied tasks: (a) scheduling time; (b) idle time.

Comparison results show that the scheduling time obtained by the proposed method is decreased about 25% less than that of BFS and about 45% less than that of WFS (Fig. 3(a) and Fig. 4(a)). Furthermore, the idle time is reduced about 75% less than that of BFS and about 85% less than that of WFS (Fig. 3(b) and Fig. 4(b)).

V. Conclusion and Future Work

This paper presented the initial results of the work on researching mapping methods for the allocation of OpenMP task graphs in underlying threads. In the scheduling phase, task-parts are placed in a global scheduling queue, then an appropriate allocation queue is selected for each task-part. In the allocation phase, the best task-part is selected from the allocation queue and executed by the relevant idle thread. The initial simulation results showed that our mapping method decreases both the scheduling time of OpenMP graph and the idle time of threads, compared to BFS and WFS. However, the difference between the results is more noticeable in the tied tasks. The main reason is that the suggested heuristics consider the essential parameters for the mapping process (e.g., execution time and response time) in the scheduling and allocation phases.

As we suggested multiple heuristics using different characteristics (e.g., execution time and response time) and the simulation results obtained by the new mapping method are inferior to those obtained by the other methods, our proposal could guarantee the predictability and robustness of the system.

Current and future work include the evaluation of the method under more complex system models, supporting the creation of multiple child tasks by a task-part at Task Scheduling Points (TSP), different graphs (in size and proportion), and different numbers of threads.

References

[1] M. Schoeberl, L. Pezzarossa, and J. Sparsø, “A multicore processor for time-critical applications,” *IEEE Des. Test*, vol. 35, no. 2, pp. 38–47, 2018.

[2] J. Hückelheim and L. Hascoët, “Source-to-Source Automatic Differentiation of OpenMP Parallel Loops,” *ACM T. Math. Software*, vol. 48, no. 1, pp. 1–32, 2022.

[3] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, and E. Quinones, “Timing characterization of OpenMP4 tasking model,” in *Proc. of Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems (CASES), Amsterdam, Netherlands*, pp. 157–166, October 4–9, 2015.

[4] A. Melani, M. A. Serrano, M. Bertogna, I. Cerutti, E. Quinones, and G. Buttazzo, “A static scheduling approach to enable safety-critical OpenMP applications,” in *Proc. of 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), Chiba, Japan*, pp. 659–665, January 16–19, 2017.

[5] J. Sun, N. Guan, X. Wang, C. Jin, and Y. Chi, “Real-Time Scheduling and Analysis of Synchronous OpenMP Task Systems with Tied Tasks,” in *Proc. of the 56th Annual Design Automation Conference 2019 (DAC ‘19), Las Vegas, USA*, pp. 1–6, June 2–6, 2019.

[6] S. Royuela, M. A. Serrano, M. Garcia-Gasulla, S. M. Bellido, J. Labarta, and E. Quinones, “The Cooperative Parallel: A Discussion About Run-Time Schedulers for Nested Parallelism,” in *Proc. of the International Workshop on OpenMP (IWOMP 2019), Auckland, New Zealand*, pp. 171–185, September 11–13, 2019.

[7] Y. Wang, X. Jiang, N. Guan, Z. Guo, X. Liu, and W. Yi, “Partitioning-Based Scheduling of OpenMP Task Systems With Tied Tasks,” *IEEE T. Parall. Distr.*, vol. 32, no. 6, pp. 1322–1339, 2020.

[8] OpenMP Architecture Review Board, “OpenMP Application Programming Interface, version 5.2,” November 2021. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf