

Task-to-Thread Mapping in OpenMP Using Fuzzy Decision Making

Mohammad Samadi, Tiago Carvalho, Luis Miguel Pinho

Polytechnic Institute of Porto & INESC TEC, Porto, Portugal; email: {mmas, tdc, lmp}@isep.ipp.pt

Sara Royuela

Barcelona Supercomputing Center, Barcelona, Spain; email: sara.royuela@bsc.es

Abstract

The performance of shared-resource multi-core hardware platforms in complex cyber-physical systems (CPSs), e.g., automotive industry, can be improved using task-based parallelism through OpenMP. However, most CPS require certain level of predictability, which challenges the efficient implementation of the task-to-thread mapping process. This exploratory work build on the fact that existing mapping methods mostly use elementary or heuristic algorithms, and the idea that artificial intelligence (AI) algorithms can be used to enhance the efficiency of such processes. Accordingly, this paper (1) evaluates the suitability of AI-based techniques in improving the performance of task-to-thread mapping in the OpenMP framework, and (2) proposes a hypothesis to perform an intelligent mapping using fuzzy logic for multi-queue schedulers to improve the predictability of the system.

Keywords: *OpenMP, predictability, task-to-thread mapping, artificial intelligence (AI), fuzzy logic.*

1 Introduction

OpenMP is a parallel framework for heterogeneous shared-memory architectures, which can be used to leverage high-performance computing (HPC) capabilities. It has been increasingly used over the past decade to improve the performance of CPSs, e.g., robotics, automation, and satellites. Although OpenMP applications can be executed on multi-core hardware platforms at high speed, the predictability (e.g., real-time analysis) of the parallel system is still a challenge.

Early versions of OpenMP support a thread-centric model for exploiting massively loop-intensive and data-parallel programs, while later versions support a task-centric model that allows complex, fine-grained, and irregular parallelism [1]. Focusing on the latter, task-to-thread mapping can be performed using centralized or distributed queues. In the former case, ready tasks are added to a single queue access by all threads to execute tasks (which increases contention in the queue) [2]. In the latter model, instead, each thread includes an individual allocation queue [3, 4], possibly reducing the contention of the system.

In parallel frameworks, task-to-thread mapping can be performed according to the timing requirements of the system. However, the main restriction of most of the existing algorithms is that they mainly use elementary or heuristic techniques that may not be able to make more efficient decisions under unforeseen conditions, which is the target of knowledge-based system [1, 2, 3, 4]. This paper explores an AI-based mechanism for task-to-thread mapping in OpenMP using fuzzy theory. A multi-queue system model is considered, where OpenMP tasks are allocated to threads' queues in the allocation phase, while they are deallocated from the queues and dispatched to threads in the dispatching phase. A fuzzy controller is proposed for the allocation phase to select the most appropriate queue for each OpenMP task. It is worth noticing that the First In, First Out (FIFO) algorithm is applied in the dispatching phase to select the task from the queue to execute by its idle thread. The main objective is to enhance the predictability of critical systems.

2 Background and Related Work

This section provides an overview of the terms needed to understand this article and introduces the most relevant state-of-the-art works related to the proposed technique.

2.1 Task-to-thread mapping

Task-to-thread mapping is a key process in parallel applications (e.g., OpenMP). It can be carried out through a single-queue (e.g., GCC) or multi-queue (e.g., LLVM) system. In the former model, after a task meets its input data dependencies, it is allocated to a queue, from which it is later deallocated and dispatched to an idle thread. In the latter model, which is considered in this paper, tasks are allocated to one of the allocations queues (belonging to threads) and then deallocated from the queue and dispatched to the thread when idle. The main problem with the former system is that all threads access the same queue for allocating and deallocating tasks, so the contention in the queue can potentially increase.

Mainstream implementations for OpenMP task-to-thread mapping use breadth-first scheduler (BFS). This algorithm creates all child task before executing them. Some implementations might also support work-first scheduler (WFS). This algorithm executes new tasks immediately after their are created, causing the suspension of the encountering (i.e., parent) task [1]. However, WFS causes in many cases the sequentialization of the execution and it is typically not used.

Melani et al. [2] have introduced two schedulers for OpenMP, including optimal and sub-optimal approaches. The optimal (but expensive) approach uses an integer linear programming (ILP) formulation to optimally allocate task-parts (i.e., the chunks of sequential code in-between task scheduling points, or points in which the scheduler might decide to suspend a task for resuming it later) to threads and minimize the task-set makespan. For the cases where the optimal approach may be too computationally expensive, the sub-optimal approach uses several tractable and less computational expensive heuristics to select tasks from a single queue and execute them by idle threads.

Samadi et al. [4] have presented a heuristic task-to-thread mapping through a multi-queue tasking system, which consists of allocation and dispatching phases. Multiple heuristics are provided in the allocation phase to select allocation queues for OpenMP tasks, as well as several heuristics are employed in the dispatching phase to select tasks from the queues and execute them by idle threads. Some heuristics are proved to reduce the application response time. However, in general, this method is still not be able to perform better than basic ones in uncertain conditions.

2.2 Artificial intelligence

AI can be used in engineering applications to solve the problems in unforeseen conditions. Some of the AI algorithms (e.g., machine learning and fuzzy logic) use predefined information (i.e., training set), while others (e.g., A* and ant colony) employ intelligent procedures (e.g., heuristic path finding) to find near-optimal solutions to complex problems [5]. AI techniques can be used in different applications (e.g., cyber-physical systems [6]) to improve the performance of the system.

Fuzzy logic is one of the most prominent AI techniques. It uses a knowledge-based system capable of making suitable decisions even in unforeseen conditions. This mechanism uses *if-then* rules composed of linguistic terms (e.g., high) as a knowledge base to first train a model based on the rules and then estimate unforeseen situations based on the model. Instead of using only boolean values, as boolean logic does, fuzzy logic uses all values between 0 and 1 [7]. All operations in this logic are performed using the fuzzy inference system (FIS), which is composed of four main components: fuzzification, rule making, inference engine, and defuzzification. Fuzzification converts a crisp value (e.g., 20°C) to a fuzzy set (e.g., [0, 0.2, 1, 0.5, 0]). Note that a universal set (e.g., [-40, -5, 5, 20, 50] °C for temperature) is defined to represent linguistic terms and crisp values as fuzzy sets. Rule making creates a fuzzy rule based on inputs and output(s) through the *if-then* rule and linguistic terms. The total fuzzy rule (i.e., model) can be built using the aggregation operation between all fuzzy rules. The inference engine predicts uncertain conditions (i.e., new output) based on new inputs and the model. Finally, defuzzification converts a fuzzy set into a crisp value [8].

3 System model

The system model considers real-time tasks implemented using graphs of OpenMP parallel computation. Each real-time

```

1 #pragma omp parallel
2 #pragma omp single
3 {
4     #pragma omp task depend(out: A,B,C)
5     Task1();
6
7     #pragma omp task depend(in: A) depend(out: D,F)
8     Task2();
9
10    #pragma omp task depend(in: C) depend(out: E)
11    Task3();
12
13    #pragma omp task depend(in: B,D,E)
14    Task4();
15
16    #pragma omp task depend(in: F)
17    Task5();
18 }
    
```

Figure 1: OpenMP data dependencies.

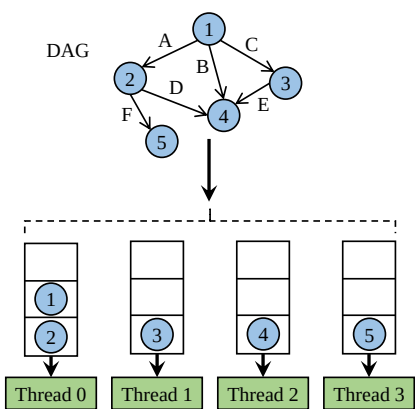


Figure 2: Multi-queue tasking systems.

task is represented as one OpenMP task dependency graph (TDG). OpenMP allows us to specify data dependencies between OpenMP tasks (Figure 1), where data dependent tasks can be executed if and only if the input data dependencies are met. ¹

In this model, a real-time task is therefore represented as a directed acyclic graph (DAG), i.e. the task dependency graph (TDG), nominated with $G = (V, E)$, where V is the set of vertices (i.e., OpenMP tasks) and E is the set of edges (i.e., the data dependencies between tasks). Furthermore, a real-time task in multi-threaded applications can be executed with multiple threads; Figure 2 exemplifies a possible deployment of the OpenMP DAG created by the code in Figure 1 onto a multi-threaded systems, where the number of OpenMP threads is assumed to be equal to the number of processor cores.

¹Note that in some of the OpenMP-based benchmarks (available at <https://gitlab.bsc.es/ampere-sw/wp2/general-information/>), e.g., the AXPY application, OpenMP tasks do not include any data dependencies, so OpenMP tasks can be allocated to the queues at the beginning of the execution process. Other cases, e.g., the Heat application, tasks can only be allocated after the data they depend upon is available. Accordingly, the complexity of an OpenMP program depends on the number of tasks and their data dependencies.

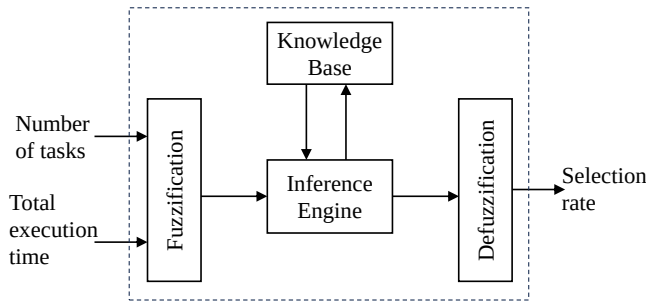


Figure 3: Structure of the suggested fuzzy controller.

4 Methodology

With the aim of achieving predictability, this work proposes a new methodology based on FIS for the allocation phase of the task-to-thread mapping process of OpenMP applications deployed on a multi-queue system. The methodology uses a fuzzy controller to select the most appropriate allocation queue for each OpenMP task when it does not include any data dependencies or they have been met. The controller, shown in Figure 3, consists of two input parameters, nominated as number of tasks and total execution time, and one output parameter, nominated as selection rate. The queue with the highest selection rate will be chosen for the task allocation process. As this controller is used for predictable OpenMP applications, the parameters have been selected in a way to improve the performance and the predictability of the system, i.e., the application's response time.

The linguistic terms used in the proposed controller are {feeble, few, normal, many, lots} for the *number of tasks*, {very small, small, mean, large, very large} for the *total execution time*, and {very low, low, medium, high, very high} for the *selection rate*. The number of (maximum) rules for this controller is based on the number of inputs, which are 5 for each possible input, hence $5^2 = 25$ in total. Then, the rules are defined in such a way that the queue with the minimum number of tasks and the least total execution time is selected in the task allocation process. For example, if the *number of tasks* is *feeble* and the *total execution time* is *very small*, then the selection rate should be *very high*. Table 1 contains some of the *if-then* rules defined in the controller.

Rule #	Number of tasks	Total execution time	Selection rate
1	feeble	very small	very high
2	few	small	high
3	normal	mean	medium
4	many	large	low
5	lots	very large	very low

Table 1: Some of the *if-then* rules used in the controller

A fuzzy set is used in FIS based on the universal set to represent a linguistic term or a crisp value. In the suggested controller, the lower and upper bounds in the universal set are {0, 4} for the *number of tasks*, {0, 100,000,000} *ns* for the *total execution time*, and {0, 1} for the *selection rate*. Consequently, the universal set for each parameter can be

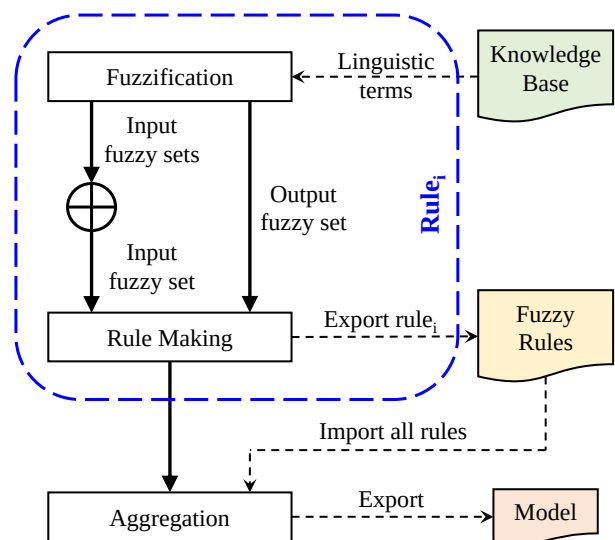


Figure 4: Workflow that builds the model using the controller.

defined with continuous numbers between the lower and upper bounds. Note that these bounds are collected based on recording the tasks-to-thread mapping using two heuristics recently proposed in the state-of-the-art [4]: (i) MNTP (selecting the most appropriate allocation queue based on the minimum number of task-parts) for the *number of tasks* parameter, and MTET (selecting the most suitable queue based on the least total execution time of tasks in the queues) for the *total execution time* parameter).

In the fuzzification process, the *triangular membership function* is used to fuzzify the linguistic terms and crisp values for the input parameters, while the *bell-shaped membership function* is used to fuzzify them for the output parameter. The *Mamdani rule-making type* (i.e., the *Maximum-Minimum function*) is used to make the fuzzy rules based on the input and output fuzzy sets for each *if-then* rule, as well as the *Maximum function* is applied to build the total fuzzy rule (i.e., the model) based on the aggregation of all the fuzzy rules. Additionally, in the defuzzification process, the *Center of Gravity* (also known as Center of Area or Centroid) method is used to defuzzify fuzzy set of the output parameter. More details about these processes can be found in recent works [8].

Figure 4 shows the process of building the model using the proposed controller. All the processes within the blue dashed region are separately performed for each fuzzy rule (i.e., *Rule_i*). First, linguistic terms belonging to the *if-then* rule in the knowledge base are converted to fuzzy sets during the fuzzification process. The input fuzzy sets are then converted to a fuzzy set to specify a unified input fuzzy set using the *Minimum function*. Finally, the fuzzy rule is created using the rule making process based on the input and output fuzzy sets. After performing this process for each *if-then* rule, all the fuzzy rules are aggregated using the aggregation process to build the final set of fuzzy rules, namely the model. This model will be used in the execution process to predict a selection rate for each queue based on new inputs.

Figure 5 illustrates the algorithm for performing an intelligent

```

1  N = Number of threads
2  NT[0..N-1] = Number of tasks
3  TET[0..N-1] = Total execution time
4  SR[0..N-1] = Selection rate
5
6  // Allocation phase
7  For each OpenMP task in the DAG do {
8    Update the information of queues
9    I = 0
10   While (I < N) {
11     NT_Fuzzy = Fuzzification (NT[I])
12     TET_Fuzzy = Fuzzification (TET[I])
13     Input_Fuzzy = Minimum (NT_Fuzzy, TET_Fuzzy)
14     SR_Fuzzy = Inference (Input_Fuzzy, Model)
15     SR[I] = Defuzzification (SR_Fuzzy)
16     I = I + 1
17   }
18   Select queue with highest selection rate
19   Allocate task to the queue
20 }
21
22 // Dispatching phase
23 For each thread do {
24   If thread is idle and its queue is not empty {
25     Select a ready task based on the FIFO
26     Dispatch task-to-thread and execute it
27   }
28 }

```

Figure 5: Intelligent task-to-thread mapping.

task-to-thread mapping in OpenMP using the suggested fuzzy controller based on the model built above. First, lines 1-to-4 define the variables. Then, during the allocation phase in lines 7-to-20, each OpenMP task with no data dependencies or whose data dependencies have been already met is allocated to one of the queues. During this process, (1) the scheduler updates the information of queues (i.e., number of tasks and total execution time), (2) a selection rate is predicted for each queue based on the new inputs and the model, and (3) the queue with the highest selection rate (to improve the timing requirements of the system) is selected, and the task is allocated to this queue. Finally, during the dispatching phase in lines 23-to-28, tasks are selected for execution following the typical implementations based on multi-queue schedulers (e.g., the LLVM compilation framework), where each idle thread selects a task from its own queue using the FIFO algorithm. Note that the approach does not use work-stealing (e.g., as implemented by LLVM) where threads can steal tasks from other threads' queues.

5 Conclusions and future work

This paper introduces the preliminary ideas to implement an OpenMP scheduler for task-to-thread mapping in multi-queue systems using a controller based the fuzzy logic. The controller includes *number of tasks* and *total execution time* as the inputs, and *selection rate* as the output to select the most suitable queue (belonging to threads) for OpenMP tasks in the allocation phase. Additionally, idle threads select ready tasks from their queues using the FIFO algorithm to execute them in the dispatching phase.

In the future work, this AI-based mapping will be simulated first and implemented later under different configurations (i.e., number of tasks, types of TDGs, and number of threads, among others) to show its performance in achieving efficient task-to-thread mapping compared to the literature.

6 Acknowledgment

This work has been partly funded by the RESPECT project (Departament de Recerca i Universitats de la Generalitat de Catalunya record No 2021 PROD 00179).

References

- [1] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, and E. Quinones, "Timing characterization of openmp4 tasking model," pp. 157–166, 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2015.
- [2] A. Melani, M. A. Serrano, M. Bertogna, I. Cerutti, E. Quinones, and G. Buttazzo, "A static scheduling approach to enable safety-critical openmp applications," pp. 659–665, 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), 2017.
- [3] M. S. Gharajeh, S. Royuela, L. M. Pinho, T. Carvalho, and E. Quinones, "Heuristic-based task-to-thread mapping in multi-core processors," pp. 1–4, 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), 2022.
- [4] M. Samadi, S. Royuela, L. M. Pinho, T. Carvalho, and E. Quinones, "Time-predictable task-to-thread mapping in multi-core processors," *Journal of Systems Architecture*, vol. 148, p. 103068, 2024.
- [5] E. Tōugu, *Algorithms and Architectures of Artificial Intelligence*, vol. 159. IOS Press, 2007.
- [6] G. Tartarisco, G. Cicceri, R. Bruschetta, A. Tonacci, S. Campisi, S. Vitabile, A. Cerasa, S. Distefano, A. Pelleggrino, P. A. Modesti, *et al.*, "An intelligent medical cyber-physical system to support heart valve disease screening and diagnosis," *Expert Systems with Applications*, vol. 238, p. 121772, 2024.
- [7] V. Novák, "Reasoning about mathematical fuzzy logic and its future," *Fuzzy Sets and Systems*, vol. 192, pp. 25–44, 2012.
- [8] M. S. Gharajeh, "Fsb-system: a detection system for fire, suffocation, and burn based on fuzzy decision making, mcdm, and rgb model in wireless sensor networks," *Wireless Personal Communications*, vol. 105, no. 4, pp. 1171–1213, 2019.