Learning to Select Scheduling Algorithms in OpenMP

Jonas H. Müller Korndörfer
jonas.korndorfer@unibe.ch
AIUB, University of Bern
Bern, Switzerland

Ali Mohammed
ali.mohammed@hpe.com
HPE HPC/AI EMEA Lab
Basel, Switzerland

Ahmed Eleliemy
ahmed.eleliemy@hpe.com
HPE HPC/AI EMEA Lab
Basel, Switzerland

Quentin Guilloteau
quentin.guilloteau@unibas.ch
INRIA
Lyon, France

Reto Krummenacher
reto.krummenacher@unibas.ch
University of Basel
Basel, Switzerland

Florina M. Ciorba
florina.ciorba@unibas.ch
University of Basel
Basel, Switzerland

1 Introduction

Modern HPC systems provide massive computational power, with increasing complexity due to parallelism and heterogeneity. This complexity makes performance highly sensitive to workload characteristics such as branching and memory access, which often cause load imbalance and performance degradation. OpenMP and more specifically alternative OpenMP libraries, such as LB4OMP [6] and others [1, 2, 8], offer a wide portfolio of scheduling algorithms to mitigate these effects. Nevertheless, selecting the most suitable algorithm for a given application-system pair constitutes an instance of Rice's algorithm selection problem [9]. Previous work, such as Auto4OMP [5], has introduced expert-based selection methods within the LB4OMP library. These rely heavily on domain knowledge and extensive experimentation, which limits their adaptability.

In this work, in an attempt to address these shortcomings, we propose a reinforcement learning (RL)-based extension to LB4OMP and Auto4OMPfor automated online selection of scheduling algorithms in OpenMP applications. We implemented and adapted the model-free `Q-Learn` and `SARSA` algorithms and conducted a comparative study against expert-based methods. The evaluation involved six applications with diverse compute- and memory-bound characteristics, executed across three systems in 720 configurations, totaling 3,600 executions. The results show that RL-based methods can effectively identify high-performing algorithms, although they incur exploration overhead, while expert-based approaches offer lightweight decisions at the risk of suboptimal selections. Combining expert knowledge with RL improves both performance and adaptability.

Contributions. This work assesses and advocates the automated selection of scheduling algorithms in OpenMP multithreaded applications to better exploit node-level parallelism. We focus on improving adaptability and performance portability using expert- and RL-based selection methods. Specifically, the contributions of this study include the following:

(1) *Proposing and implementing* RL-based scheduling algorithm selection as an add-on framework integrated to the LB4OMP library for performance evaluation.
(2) *Conducting a large-scale performance analysis* to assess the strengths and limitations of expert- and RL-based selection methods.
(3) *Providing recommendations* on scenarios where each method excels and strategies to integrate expert knowledge with RL for enhanced performance.

2 Automated Scheduling Algorithm Selection

In this study, we consider two categories of automated selection methods: expert-based (from Auto4OMP [5]) and RL-based (proposed in this work). Expert-based approaches, such as `RandomSel`, `ExhaustiveSel`, and `ExpertSel` [5], rely on predefined rules to guide scheduling algorithm choice. These methods minimize exploration overhead yet they may miss optimal strategies when workloads change. In contrast, model-free RL-based approaches as the ones proposed in this work, specifically `Q-Learn`and `SARSA`, explore and adapt to loop behavior over time, learning from execution feedback to improve scheduling decisions.

The RL-based methods were implemented into an external framework "kmp_agent_provider.h" and integrated with LB4OMP for performance evaluation. The "kmp_agent_provider.h" framework can be reused with minimal modifications in any scheduling library. Figure 1 illustrates this integration, RL agents operate with the existing OpenMP scheduling infrastructure to select from a portfolio of 12 algorithms, including those from the OpenMP standard, LLVM runtime, and LB4OMP. These agents rely on loop-level identifiers to treat each loop instance independently, and their decisions are driven by reward functions based on loop execution time (LT) and load imbalance (LIB). The RL-based methods follow an 'explore-first policy' to ensures all scheduling algorithms are initially tested, after which learned policies refine the selection. This design enables OpenMP applications to benefit from automated algorithm selection while serving as a platform for further improvements such as hybrid expert–RL strategies.

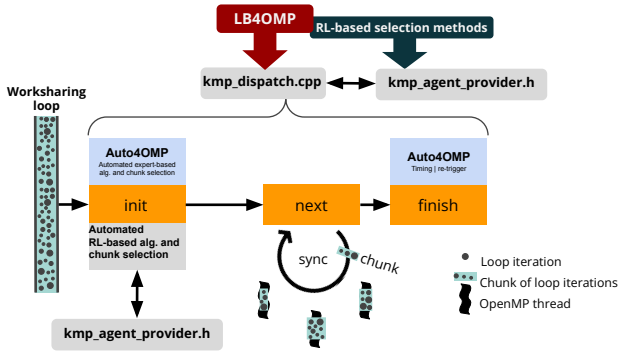

Figure 1: Scheduling in LB4OMP augmented with RL-based selection of scheduling algorithms.

Table 1: Design of factorial experiments, resulting in 720 different configurations and a total of 3'600 executions.

Factors		Values		Properties
Applications		Mandelbrot		**Compute-bound kernel** N = 262, 144 iterations| T = 500 time-steps | Total loops = 3 | Modified loops = 3 L0 = constant workload imbalance, L1 = increasing workload imbalance, L3 = decreasing workload imbalance
		STREAM Triad		**Memory-bound Benchmark** N = 2, 000, 000, 000 iterations | T = 500 time-steps | Total loops = 1 | Modified loops = 1 L0 = no workload imbalance
		Triangle Counting (TC)		**Graph theory and network analysis kernel** N = 1, 048, 576 iterations | T = 500 time-steps | Total loops = 1 | Modified loops = 1 L0 = highly imbalanced due to sparse input
		HACCKernels		**Cosmology N-body code, HACC's particle force kernels** N = 600, 000 iterations | T = 500 time-steps | Total loops = 3 | Modified loops = 1 L0 = no workload imbalance
		LULESH		**Computational fluid dynamics miniapp** N = 21, 952, 000 iterations | T = 500 time-steps | Total loops = 39 | Modified loops = 4 L0, L1, L2, L3 = mildly load imbalanced
		SPHYNX Evrard collapse		**N-body and hydrodynamics simulation scientific application** N = 1, 000, 000 particles (iterations) | T = 500 time-steps | Total loops = 37 | Modified loops = 1 L0 = variable workload imbalance
Loop Scheduling	**OpenMP standard** *(also in LB4OMP)*	`STATIC`		Straightforward parallelization
		`SS` , `GSS`		Dynamic and non-adaptive self-scheduling algorithms.
	LB4OMP library [6]	`mFAC2`, `Auto(LLVM)`(LLVM `schedule(auto)`), `TSS`, `Static Steal`		
		`AWF-B`, `AWF-C`, `AWF-D`, `AWF-E`, `mAF`		Dynamic and adaptive self-scheduling algorithms.
Scheduling Algorithm Selection	**Manual**	Manual Selection	`Oracle`*	Ideal combination of the best choices of scheduling algorithm selection across application loops, and time-steps. Selection based on exhaustive experimentation of all scheduling algorithms.
	Automated from the LB4OMP library	**Auto4OMP extension** [5]	`RandomSel`* `ExhaustiveSel`* `ExpertSel`*	Expert-based scheduling algorithm selection across application loops and time-steps.
		RL4OMP extension	`Q-Learn`* `SARSA`*	RL-based scheduling algorithm selection across application loops and time-steps.
Chunk parameter		Default		Chunk size $=N/P$ for static and 1 for all other scheduling algorithms.
		`expChunk` [5]		Expert chunk parameter: "a point at $\frac{1}{Golden\ ratio(1.618)} = 0.618$ on the curve between $N/(iP)$ and 1, with i increasing in steps of 2" [5].
RL general configuration		α=0.5, γ=0.5, reward values: positive 0.01, neutral -2.0, negative -4.0		Variables required to configure the RL selection process.
RL initializer		`zero`		Starts all possible actions with a reward value of 0.
RL exploration policy		`explore-first`		Selects every combination of scheduling algorithms once at the beginning of the learning process. Requires 144 time-steps to learn and select the first scheduling algorithm.
RL reward type		`LT` `LIB`		Loop time reward. Load imbalance reward.
Computing nodes		Broadwell		Intel Xeon E5-2640 v4 (2 sockets, 10 cores each) P = 20 without hyperthreading, Pinning: `OMP_PLACES` = cores `OMP_PROC_BIND` = close
		Cascade-Lake		Intel Xeon Gold 6258R (2 sockets, 28 cores each) P = 56 without hyperthreading, Pinning: `OMP_PLACES` = cores `OMP_PROC_BIND` = close
		EPYC		AMD EPYC 7742 (2 sockets, 64 cores each) P = 128 without hyperthreading, Pinning: `OMP_PLACES` = cores `OMP_PROC_BIND` = close

Metrics	T_{par}^{loop}	Parallel loop execution time (s).
	$execution\ imbalance = \frac{maximum\ time - average\ time}{maximum\ time} \times \frac{P}{P-1}$	Execution imbalance (%).

* Selects a scheduling algorithm among: {`STATIC`, `SS`, `GSS`, `Auto(LLVM)`, `TSS`, `Static Steal`, `mFAC2`, `AWF-B`, `AWF-C`, `AWF-D`, `AWF-E`, and `mAF`}.
Text in orange highlights the expert-based selection methods and expert chunk parameter proposed in prior work [5].
The text in blue denotes the novel contributions of this work and highlights configurations that affect only RL-based selection methods.

3 Experiments and Analysis

We designed a factorial set of experiments covering six applications, three systems, and multiple configuration parameters, resulting in a total of 3,600 executions, see Table 1. In the poster and here we show an overview of the results and select one application (STREAM Triad) to dive in details. T denotes time-steps, P number of threads, N iterations, and L, IDs of modified loops that use LB4OMP. Full set of results at [7] and more details in [3].

For performance evaluation, we introduce the concept of a `Oracle`. It represents a *manually* selected set of scheduling algorithms and chunk parameters (default or `expChunk` [5]) optimized for peak performance across all combinations of application, loop, time-step, and system. `Oracle` is derived from the experiments with each of the 12 scheduling algorithms in the portfolio.

Figure 2 summarizes the comparative performance of all methods against the `Oracle`. The heatmap highlights that RL-based approaches, when combined with expert knowledge such as the `expChunk` parameter, significantly reduce performance degradation compared to `Oracle` across applications. For example, SPHYNX Evrard collapse and STREAM Triad achieved over an order-of-magnitude improvement when expert guidance was integrated with RL. These results demonstrate that hybrid methods, blending exploration-driven RL with targeted expert insights, offer the best balance between adaptability and efficiency in OpenMP scheduling.

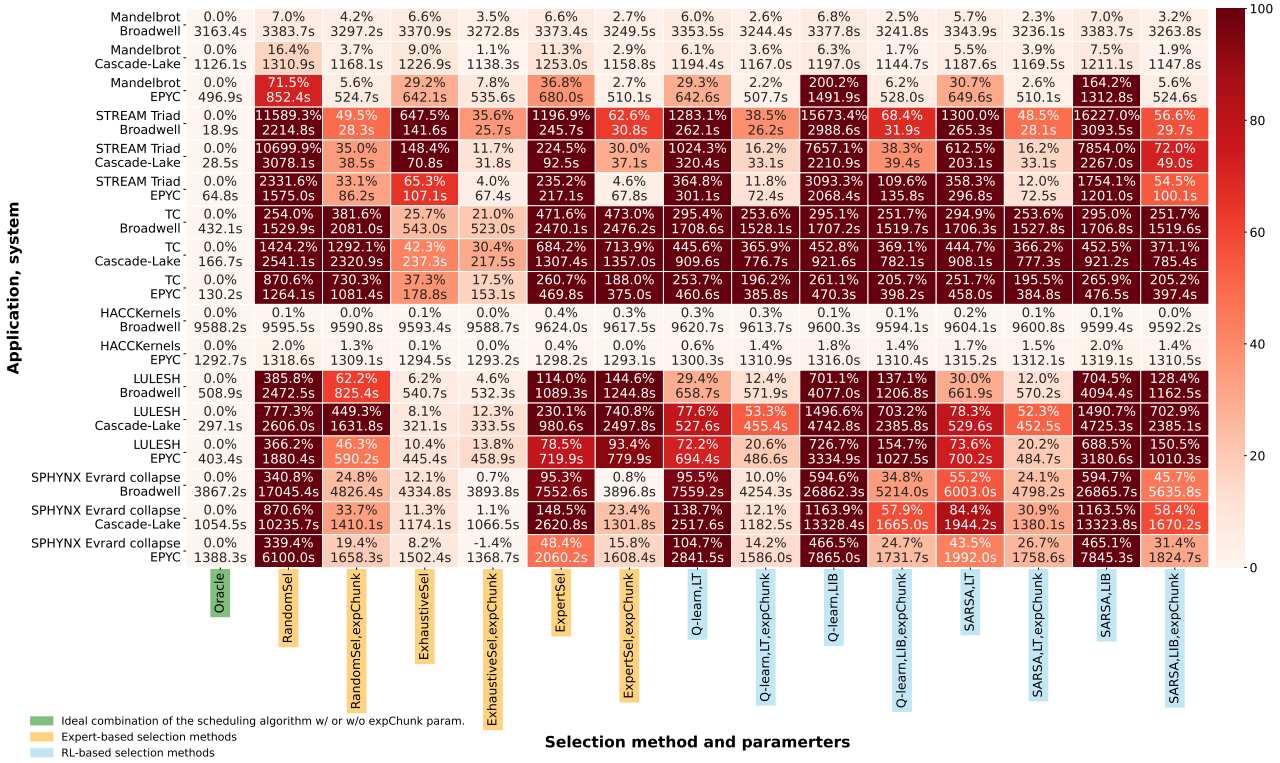

Figure 2: Performance degradation (%) of expert- vs. RL-based scheduling relative to `Oracle`, across application–system pairs (darker = higher loss).

4 Conclusion and Future Work

In this work, we present a comparative study of expert- and RL-based approaches for automated OpenMP scheduling algorithm selection. We propose the use of two RL-based methods (`Q-Learn`and `SARSA`) and evaluate them against two expert-based (`ExhaustiveSel` and `ExpertSel` [5]) ones across six applications and three systems. We demonstrates that no single scheduling algorithm or selection strategy consistently achieves optimal performance across diverse applications and systems. Expert-based methods deliver low-overhead and stable performance, while RL-based methods provide greater adaptability at the cost of exploration overhead. Combining expert knowledge with RL, e.g., via the `expChunk` parameter, substantially reduces learning costs and improves accuracy, stressing the potential of hybrid approaches. Looking forward, reducing RL exploration overhead through model-based [4] or transfer learning techniques [10], validating results on emerging hardware, and extending selection methods to heterogeneous and multilevel scheduling contexts (e.g., CPUs, GPUs, and MPI) are critical directions. Ultimately, the "no-free-lunch" nature of the scheduling problem [11] highlights the need for adaptive, data-driven methods that can dynamically balance efficiency, portability, and robustness across evolving HPC systems.

References

[1] F. M. Ciorba, C. Iwainsky, and P. Buder. 2018. OpenMP Loop Scheduling Revisited: Making a Case for More Schedules. In *Evolving OpenMP for Evolving Architectures: 14th International Workshop on OpenMP, IWOMP*. 21–36.

[2] Franziska Kasielke, Ronny Tschüter, Christian Iwainsky, Markus Velten, Florina M. Ciorba, and Ioana Banicescu. 2019. Exploring Loop Scheduling Enhancements in OpenMP: An LLVM Case Study. In *P. Intern. Symp. on Par. Dist. Comp.* Amsterdam.

[3] Jonas H. Müller Korndörfer, Ali Mohammed, Ahmed Eleliemy, Quentin Guilloteau, Reto Krummenacher, and Florina M. Ciorba. 2025. A Comparative Study of OpenMP Scheduling Algorithm Selection Strategies. arXiv:2507.20312 [cs.DC] https://arxiv.org/abs/2507.20312

[4] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. 2023. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning* 16, 1 (2023), 1–118.

[5] Ali Mohammed, Jonas H. Müller Korndörfer, Ahmed Eleliemy, and Florina M. Ciorba. 2022. Automated Scheduling Algorithm Selection and Chunk Parameter Calculation in OpenMP. *IEEE Transactions on Parallel and Distributed Systems* 33 (2022), 12.

[6] Jonas H. Müller Korndörder, Ahmed Eleliemy, Ali Mohammed, and Florina M. Ciorba. 2022. LB4OMP: A Dynamic Load Balancing Library for Multithreaded Applications. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2022), 830–841.

[7] Jonas H. Müller Korndörfer, Ali Mohammed, Ahmed Eleliemy, Quentin Guilloteau, Reto Krummenacher, and Florina M. Ciorba. 2024. Data, scripts, and plots for the paper "A Comparative Study of OpenMP Scheduling Algorithm Selection Strategies". doi:10.5281/zenodo.13255332

[8] Pedro Henrique Penna, Antônio Tadeu A. Gomes, Márcio Castro, Patricia DM Plentz, Henrique C. Freitas, François Broquedis, and Jean-François Méhaut. 2019. A Comprehensive Performance Evaluation of the BinLPT Workload-aware Loop Scheduler. *J. Concurrency & Computation: Practice & Experience* (2019).

[9] John R. Rice. 1976. The Algorithm Selection Problem. In *Advances in Computers*, Morris Rubinoff and Marshall C. Yovits (Eds.). Advances in Computers, Vol. 15. Elsevier, 65–118. doi:10.1016/S0065-2458(08)60520-3

[10] Lisa Torrey and Jude Shavlik. 2010. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 242–264.

[11] Xin-She Yang, Xing-Shi He, and Qin-Wei Fan. 2020. Chapter 7 - Mathematical framework for algorithm analysis. In *Nature-Inspired Computation and Swarm Intelligence*, Xin-She Yang (Ed.). Academic Press, 89–108.