

OpenMP[®]

SC24 Booth Talk Series



Toward Smart Scheduling in OpenMP

J. H. Müller Korndörfer[‡], A. Mohammed[†], A. Eleliemy[†], Q. Guilloteau[‡] and F. M. Ciorba[‡]

[‡]Department of Mathematics and Computer Science, University of Basel, Switzerland

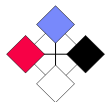
[†]HPE HPC/AI EMEA Research Lab (ERL), Switzerland

Toward Smart Scheduling in OpenMP

J. H. Müller Korndörfer[‡], A. Mohammed[†], A. Eleliemy[†], Q. Guilloteau[‡] and F. M. Ciorba[‡]

[‡]Department of Mathematics and Computer Science, University of Basel, Switzerland

[†]HPE HPC/AI EMEA Research Lab (ERL), Switzerland



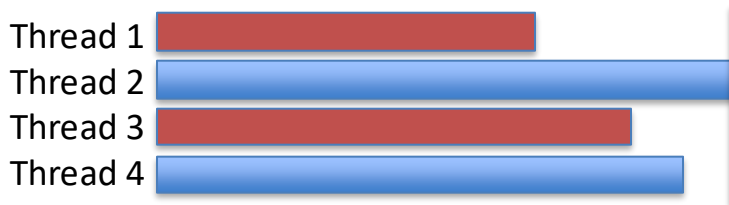
DAPHNE

Outline

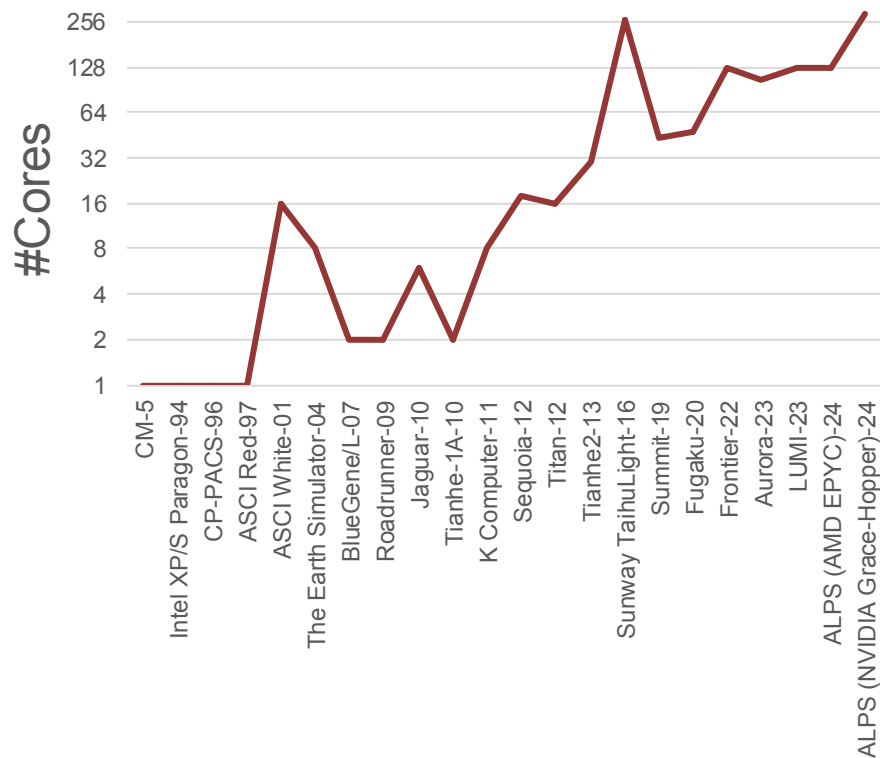
- Need for Smart Scheduling in OpenMP
- Toward Smart Scheduling in OpenMP via
 - Expert-based Automated Scheduling Algorithm Selection
 - RL-based Automated Scheduling Algorithm Selection
- Results: Expert-based vs. RL-based Selection
- Take Away's

Performance Degradation due to Load Imbalance

- Increasing #cores/node
- Increased complexity to parallelize and exploit available computing power
- Load imbalance degrades performance, scalability, and wastes energy

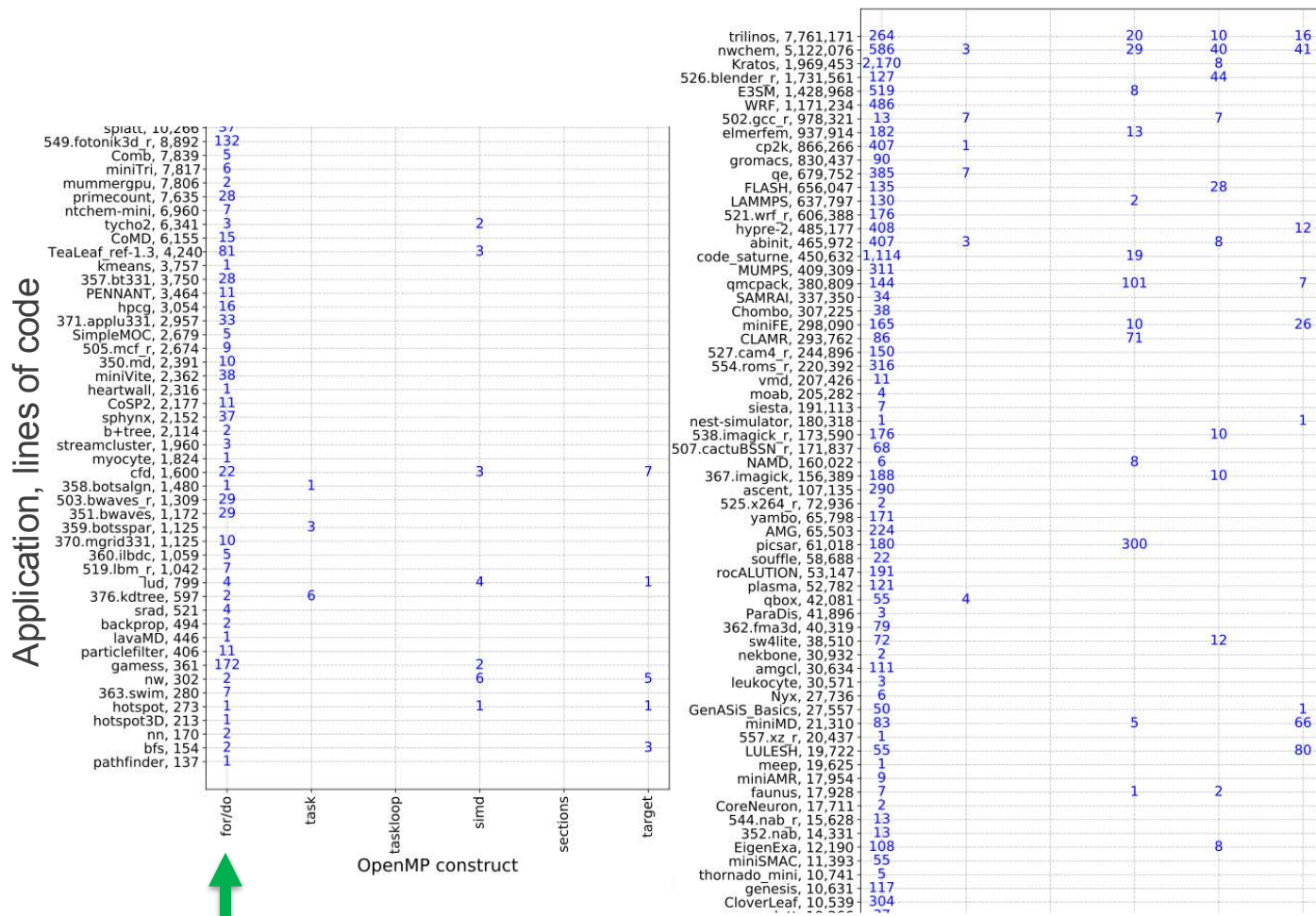


#Cores/node in top systems
on the Top500 list



Impact of Optimizing OpenMP Parallel for/do Constructs

- **111 Applications** (C, C++, Fortran). Sources: SPEC (CPU/OMP), CORAL2, Mantevo, Rodinia benchmarks, and several large-scale scientific applications available on GitHub
- **OpenMP features searched:**
 - parallel for/do
 - task, taskloop
 - simd
 - sections
 - target
- **Most applications rely on parallel OpenMP for/do loops** for shared memory parallelism



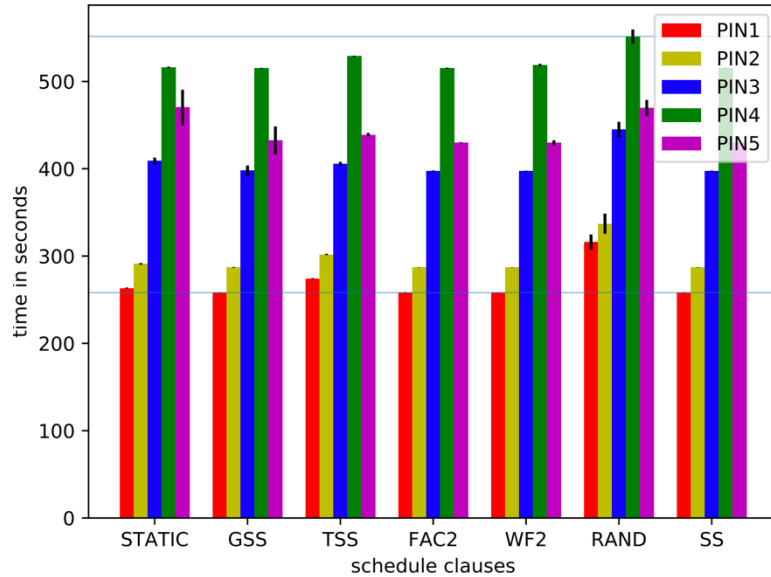
Scheduling of Worksharing (for) Loops in OpenMP

static (default)
dynamic runtime
guided auto

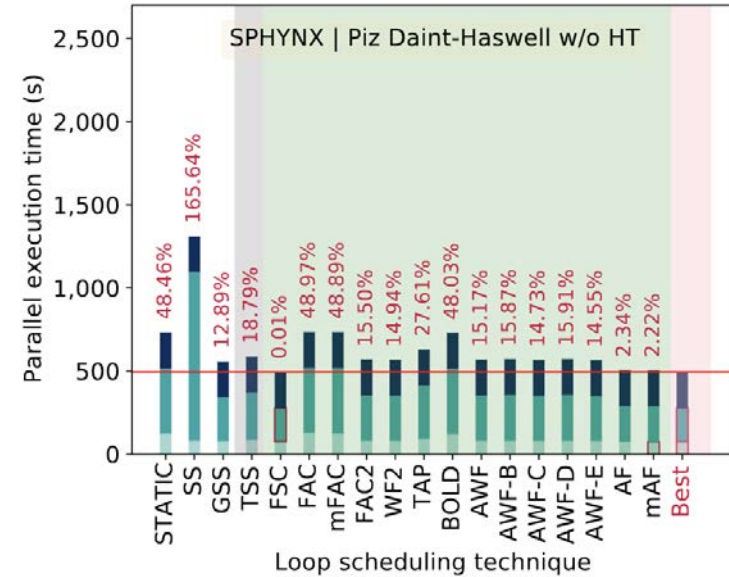
#iteration assigned/round → static/dynamic
threshold for #iteration assigned → guided

```
#pragma omp parallel for schedule(kind, chunk)
for(i = 0; i < size; i++)
{
    computations ...
}
```

Making a Case for More Efficient Scheduling Algorithms in OpenMP



F. M. Ciorba, C. Iwainsky, and P. Buder, “**OpenMP Loop Scheduling Revisited: Making a Case for More Schedules**”
iWomp 2018, Barcelona, Spain, September 2018.



J. H. Müller Korndörfer, A. Eleliemy, A. Mohammed, F. M. Ciorba. “**LB4OMP: A Dynamic Load Balancing Library for Multithreaded Applications**”. TPDS, August 2021.



“Decision Paralysis”: Which Scheduling Algorithm is Most Efficient?

- **Many** choices for OpenMP `schedule(kind)`
- **Too many** values for OpenMP `chunk`
- Tuples of scheduling choices needed
 - per loop
 - per time-step
 - per application
 - per system

The schedule kind **auto** allows* an OpenMP implementation to choose any possible mapping of iterations in a loop construct to threads in the team.

* According to OpenMP Specification 5.2

Toward Smart Scheduling in OpenMP via Automatic Algorithm Selection

For a problem instance $x \in P$, with features $f(x) \in F$, **find** the **selection mapping** $S(f(x))$ into algorithm space A , such that the selected algorithm $\alpha \in A$ maximizes performance $y(\alpha(x)) \in Y$ when applied.

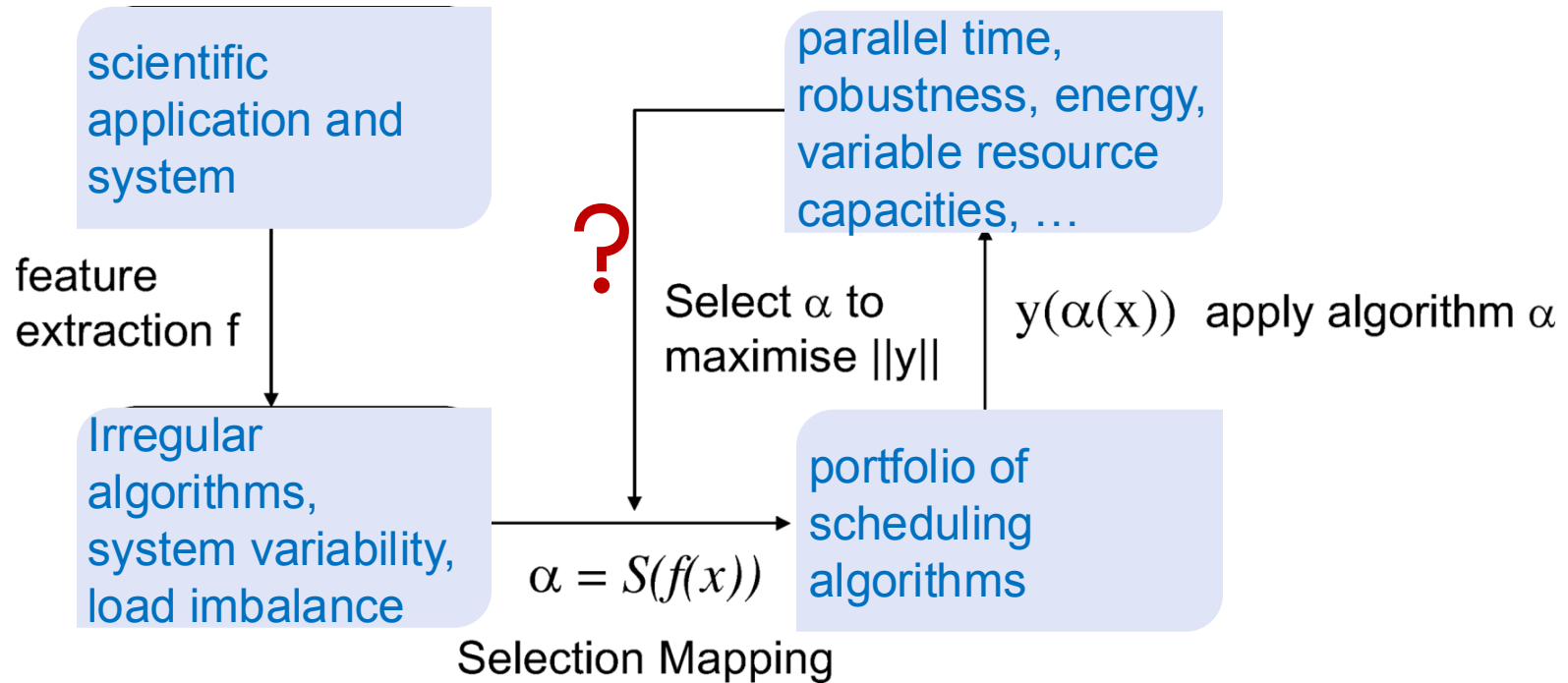


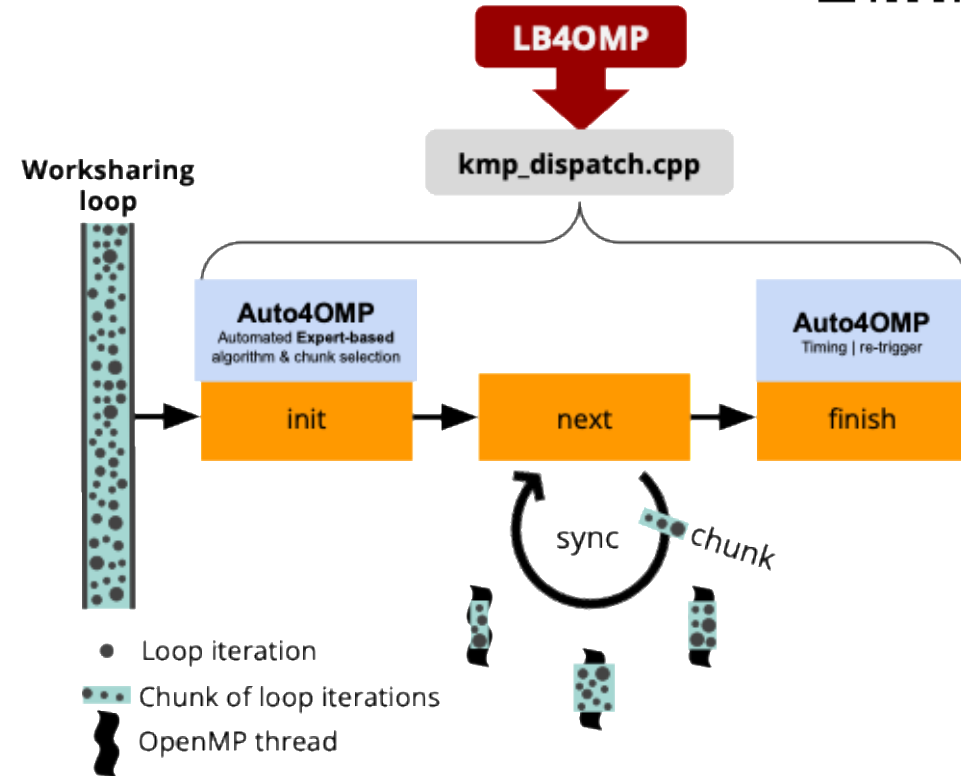
Figure from [Smith-Miles, 2008] Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. 41(1)
[Rice, 1976] "The algorithm selection problem," in Advances in Computers, 1976, pp. 65–118

Expert-based Automated Scheduling Algorithm Selection

<http://github.com/unibas-dmi-hpc/LB4OMP>



- **Leverages** `auto` as a scheduling option for `schedule(kind)`
- **3 expert-based** scheduling algorithm selection methods
 - RandomSel
 - ExhaustiveSel
 - ExpertSel
- **Selects** from a portfolio of scheduling algorithms
 - STATIC, SS, GSS, GAC, TSS, Static Steal, mFAC2, AWF-B, AWF-C, AWF-D, AWF-E, mAF



A. Mohammed, J. H. Müller Korndörfer, A. Eleliemy, F. M. Ciorba. "Automated Scheduling Algorithm Selection and Chunk Parameter Calculation in OpenMP". TPDS, December 2022.



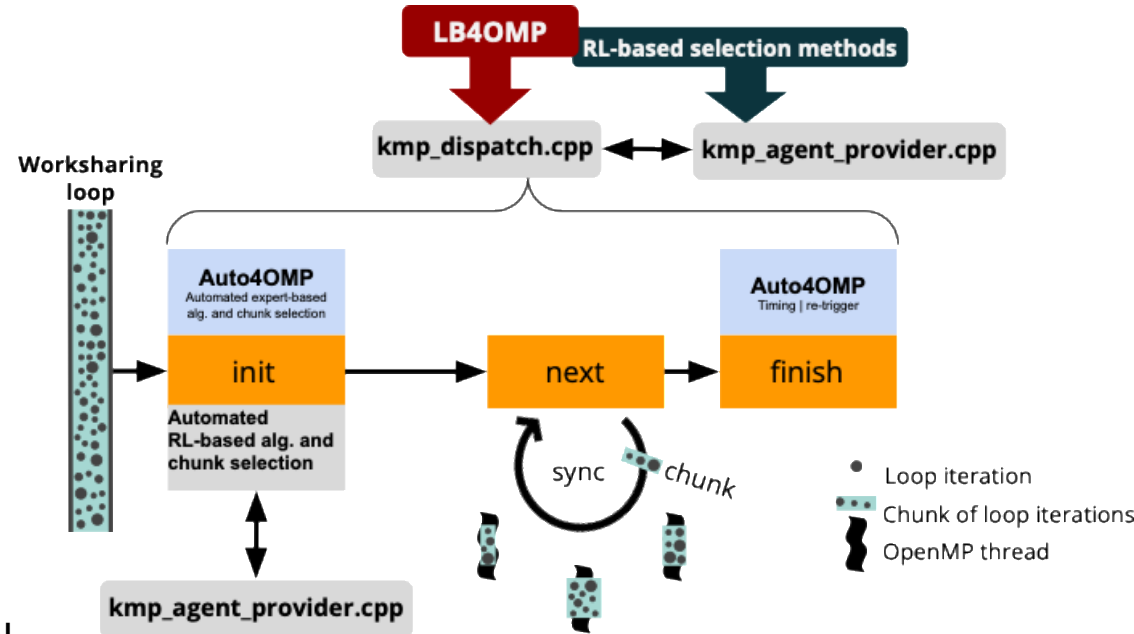
- **Extends** the LLVM OpenMP runtime library with the `expert chunk parameter`

RL-based Automated Scheduling Algorithm Selection

<http://github.com/unibas-dmi-hpc/LB4OMP>



- **Leverages** `auto` as a scheduling option for `schedule(kind)`
- **2 RL-based** scheduling algorithm selection methods
 - Q-learn
 - Sarsa
- **2 reward** types
 - Loop execution time (LT)
 - Loop load imbalance (LIB)
- **Selects** from a portfolio of scheduling algorithms
 - STATIC, SS, GSS, GAC, TSS, Static Steal, mFAC2, AWF-B, AWF-C, AWF-D, AWF-E, mAF



Results: Comparison between Expert-based vs. RL-based selection

Mandelbrot (3 loops)

SPHYNX Evrard Collapse (1 loop)

Stream Triad (1 loops)

Triangle counting (1 loop)

HACCKernels (1 loop)

Lulesh (4 loops)

miniHPC: Intel 2x10 cores Broadwell,
2x28 cores CascadeLake, and **2x32
cores AMD EPYC**

Loop execution time (LT)

Load imbalance (LIB)

Change of loop execution time

Change of load imbalance

Oracle
(ground truth)

Auto4OMP:
RandomSel
ExhaustiveSel
ExpertSel

RL4OMP:
Q-Learn
SARSA

Default chunk
Expert chunk

Parallel execution time
Performance variation

Select α to
minimise $\|y\|$

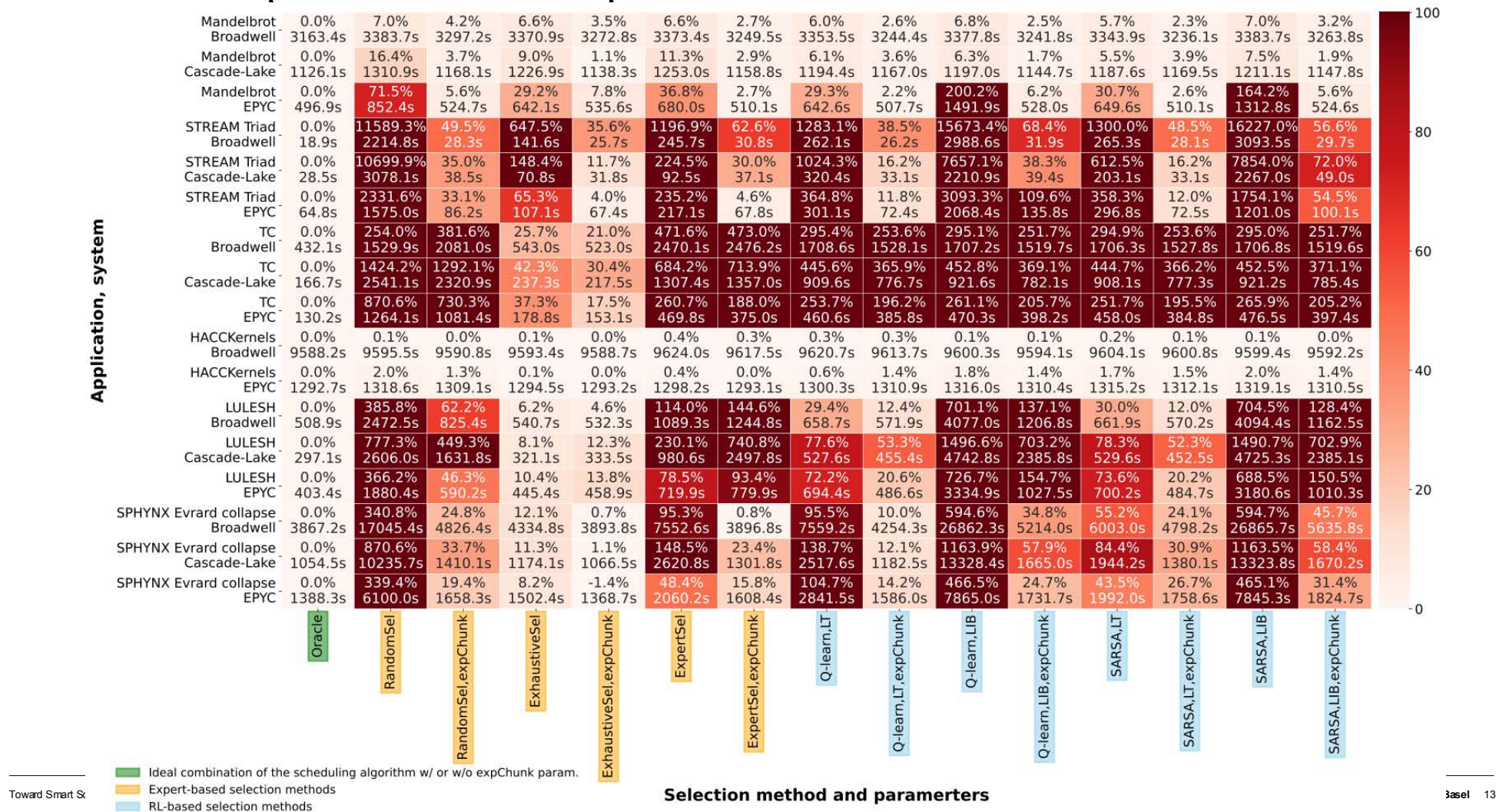
$y(\alpha(x))$ apply algorithm α

STATIC, SS, GSS, GAC
(LLVM auto), TSS, Static
Steal, mFAC2, AWF-B, AWF-
C, AWF-D, AWF-E, mAF

720 configurations
x 5 repetitions
**= 3'600 total
experiments**

Selection Mapping

Results: Comparison between Expert-based vs. RL-based selection



Results: Comparison between Expert-based vs. RL-based selection

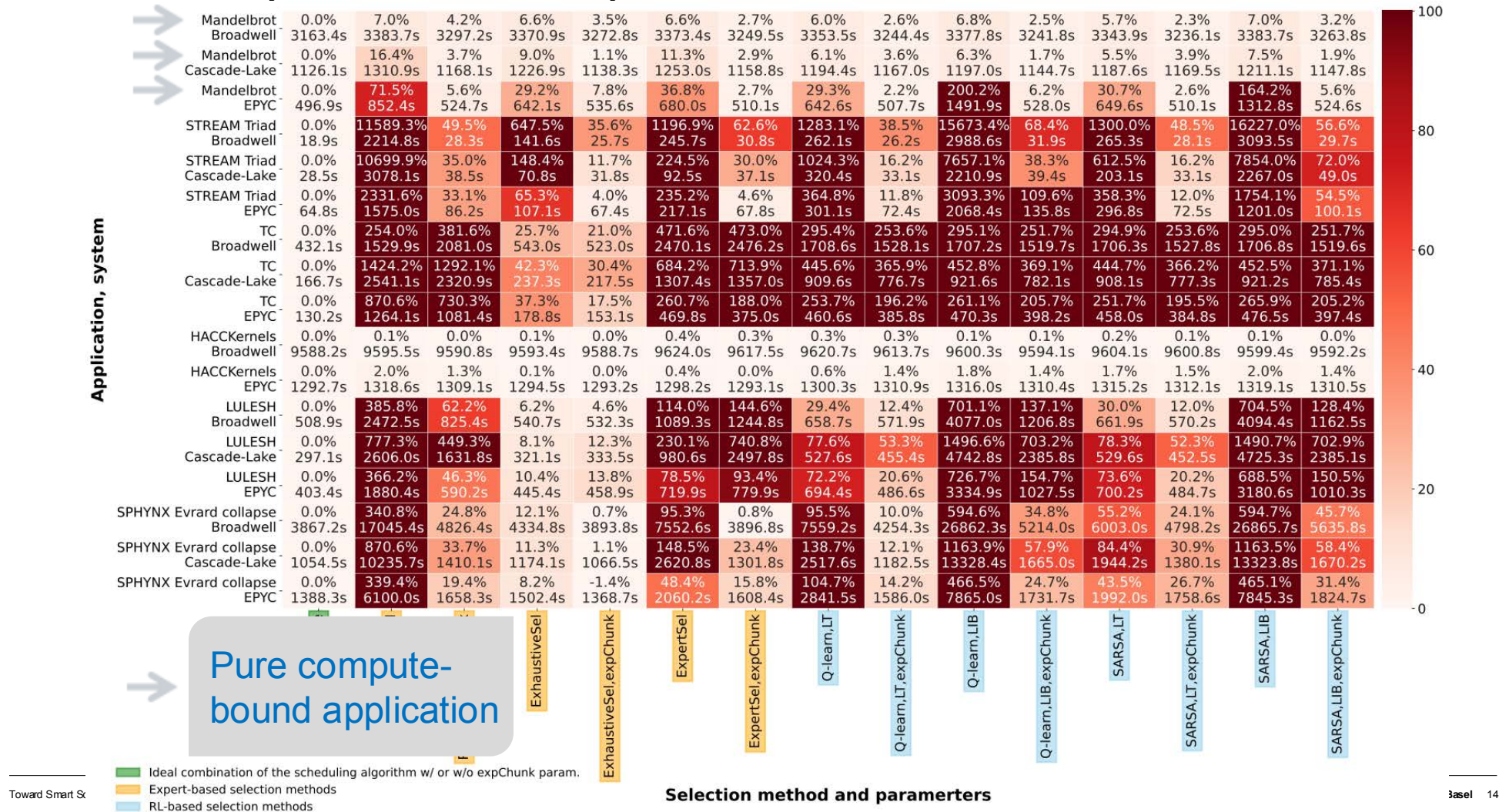


Figure 10 is a heatmap illustrating the performance of various applications across different selection methods and parameters. The applications are listed on the left, and the selection methods and parameters are listed on the top. The color scale on the right indicates the performance value, ranging from 0 (dark red) to 100 (dark blue).

The applications are categorized into three groups based on their selection method:

- Pure compute-bound application:** Mandelbrot, Cascade-Lake, STREAM Triad, EPYC, TC, HACC, LULESH, SPHYNX.
- Compute- and memory-bound application:** Q-learn, LIB, expChunk, SARSA, LT, expChunk, LIB, expChunk.

The selection methods and parameters are:

- Selection method:** Ideal combination of the scheduling algorithm w/ or w/o expChunk param., Expert-based selection methods, RL-based selection methods.
- Parameters:** Q-learn, LIB, expChunk, SARSA, LT, expChunk, LIB, expChunk.

The heatmap shows that performance is generally higher for pure compute-bound applications and lower for compute- and memory-bound applications. The color scale indicates that performance values range from 0 to 100.

Results: Comparison between Expert-based vs. RL-based selection

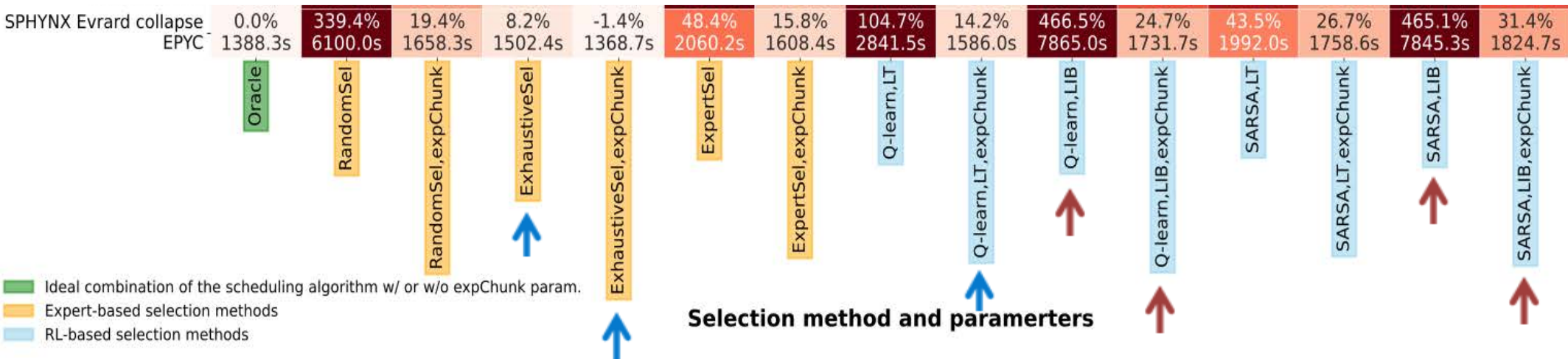
Application, system																	
		0.0%	7.0%	4.2%	6.6%	3.5%	6.6%	2.7%	6.0%	2.6%	6.8%	2.5%	5.7%	2.3%	7.0%	3.2%	
Mandelbrot	Broadwell	3163.4s	3383.7s	3297.2s	3370.9s	3272.8s	3373.4s	3249.5s	3353.5s	3244.4s	3377.8s	3241.8s	3343.9s	3236.1s	3383.7s	3263.8s	
	Cascade-Lake	1126.1s	1310.9s	1168.1s	1226.9s	1138.3s	1253.0s	1158.8s	1194.4s	1167.0s	1197.0s	1144.7s	1187.6s	1169.5s	1211.1s	1147.8s	
Mandelbrot	EPYC	496.9s	852.4s	524.7s	642.1s	535.6s	680.0s	510.1s	642.6s	507.7s	1491.9s	528.0s	649.6s	510.1s	1312.8s	524.6s	
	EPYC	496.9s	852.4s	524.7s	642.1s	535.6s	680.0s	510.1s	642.6s	507.7s	1491.9s	528.0s	649.6s	510.1s	1312.8s	524.6s	
STREAM Triad	Broadwell	18.9s	11589.3%	49.5%	647.5%	35.6%	1196.9%	62.6%	1283.1%	38.5%	15673.4%	68.4%	1300.0%	48.5%	16227.0%	56.6%	
	Cascade-Lake	28.5s	10699.9%	35.0%	148.4%	11.7%	224.5%	30.0%	1024.3%	16.2%	7657.1%	38.3%	612.5%	16.2%	7854.0%	72.0%	
STREAM Triad	EPYC	64.8s	2331.6%	33.1%	65.3%	4.0%	235.2%	4.6%	364.8%	11.8%	3093.3%	109.6%	358.3%	12.0%	1754.1%	54.5%	
	EPYC	64.8s	1575.0s	86.2s	107.1s	67.4s	217.1s	67.8s	301.1s	72.4s	2068.4s	135.8s	296.8s	72.5s	1201.0s	100.1s	
TC	Broadwell	432.1s	1529.9s	2081.0s	543.0s	523.0s	2470.1s	2476.2s	1708.6s	1528.1s	1707.2s	1519.7s	1706.3s	1527.8s	1706.8s	1519.6s	
	Cascade-Lake	166.7s	1424.2%	1292.1%	42.3%	30.4%	684.2%	713.9%	445.6%	365.9%	452.8%	369.1%	444.7%	366.2%	452.5%	371.1%	
TC	EPYC	130.2s	870.6%	730.3%	37.3%	17.5%	260.7%	188.0%	253.7%	196.2%	261.1%	205.7%	251.7%	195.5%	265.9%	205.2%	
	EPYC	130.2s	1264.1s	1081.4s	178.8s	153.1s	469.8s	375.0s	460.6s	385.8s	470.3s	398.2s	458.0s	384.8s	476.5s	397.4s	
HACKernels	Broadwell	9588.2s	9595.5s	9590.8s	9593.4s	9588.7s	9624.0s	9617.5s	9620.7s	9613.7s	9600.3s	9594.1s	9604.1s	9600.8s	9599.4s	9592.2s	
	EPYC	1292.7s	2.0%	1.3%	0.1%	0.0%	0.4%	0.0%	0.6%	1.4%	1.8%	1.4%	1.7%	1.5%	2.0%	1.4%	
LULESH	Broadwell	508.9s	385.8%	62.2%	6.2%	4.6%	114.0%	144.6%	29.4%	12.4%	701.1%	137.1%	30.0%	12.0%	704.5%	128.4%	
	Cascade-Lake	297.1s	777.3%	449.3%	8.1%	12.3%	230.1%	740.8%	77.6%	53.3%	1496.6%	703.2%	78.3%	52.3%	1490.7%	702.9%	
LULESH	EPYC	403.4s	366.2%	46.3%	10.4%	13.8%	78.5%	93.4%	72.2%	20.6%	726.7%	154.7%	73.6%	20.2%	688.5%	150.5%	
	EPYC	403.4s	1880.4s	590.2s	445.4s	458.9s	719.9s	779.9s	694.4s	486.6s	3334.9s	1027.5s	700.2s	484.7s	3180.6s	1010.3s	
SPHYNX Evrard collapse	Broadwell	3867.2s	340.8%	24.8%	12.1%	0.7%	95.3%	0.8%	95.5%	10.0%	594.6%	34.8%	55.2%	24.1%	594.7%	45.7%	
	Cascade-Lake	1054.5s	870.6%	33.7%	11.3%	1.1%	148.5%	23.4%	138.7%	12.1%	1163.9%	57.9%	84.4%	30.9%	1163.5%	58.4%	
SPHYNX Evrard collapse	EPYC	1388.3s	339.4%	19.4%	8.2%	-1.4%	48.4%	14.8%	104.7%	14.2%	466.5%	24.7%	43.5%	26.7%	465.1%	31.4%	
	EPYC	1388.3s	6100.0s	1658.3s	1502.4s	1368.7s	2060.2s	1608.4s	2841.5s	1586.0s	7865.0s	1731.7s	1992.0s	1758.6s	7845.3s	1824.7s	

Pure compute-bound application

Compute- and memory-bound application

Pure memory-bound application

Results: Comparison between Expert-based vs. RL-based selection

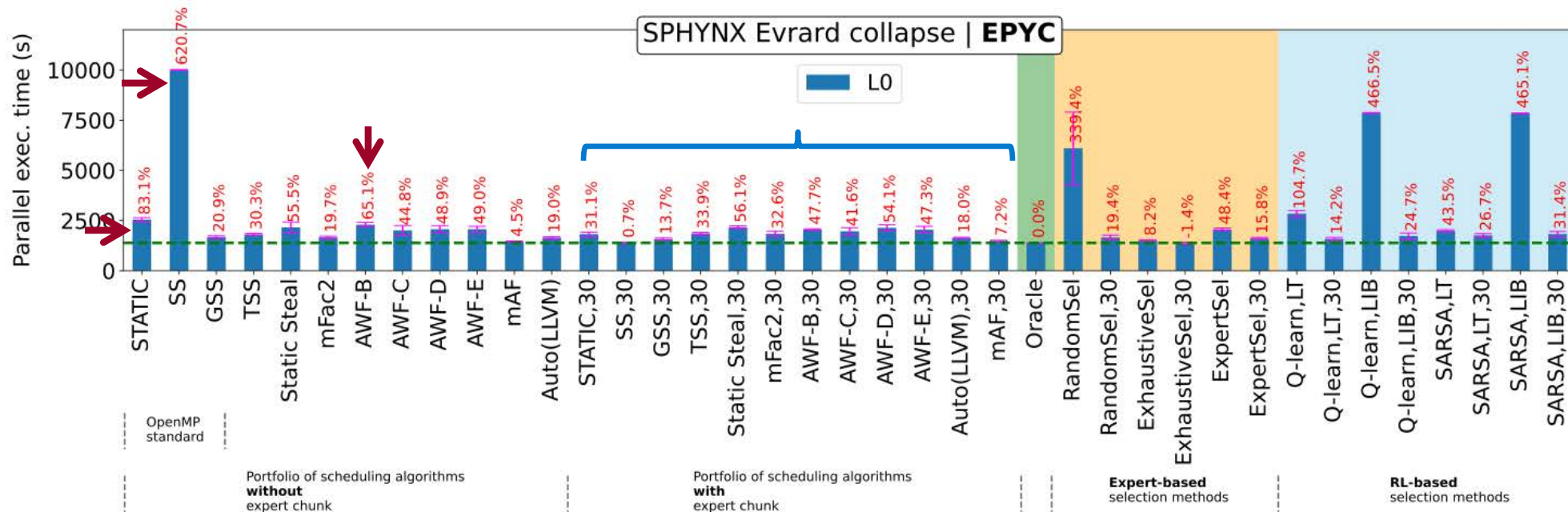


→ Highest performing **expert-based** (and overall) method

→ Highest performing **RL-based** (and overall) method

→ LIB reward achieves low performance

Results: Comparison between Expert-based vs. RL-based selection

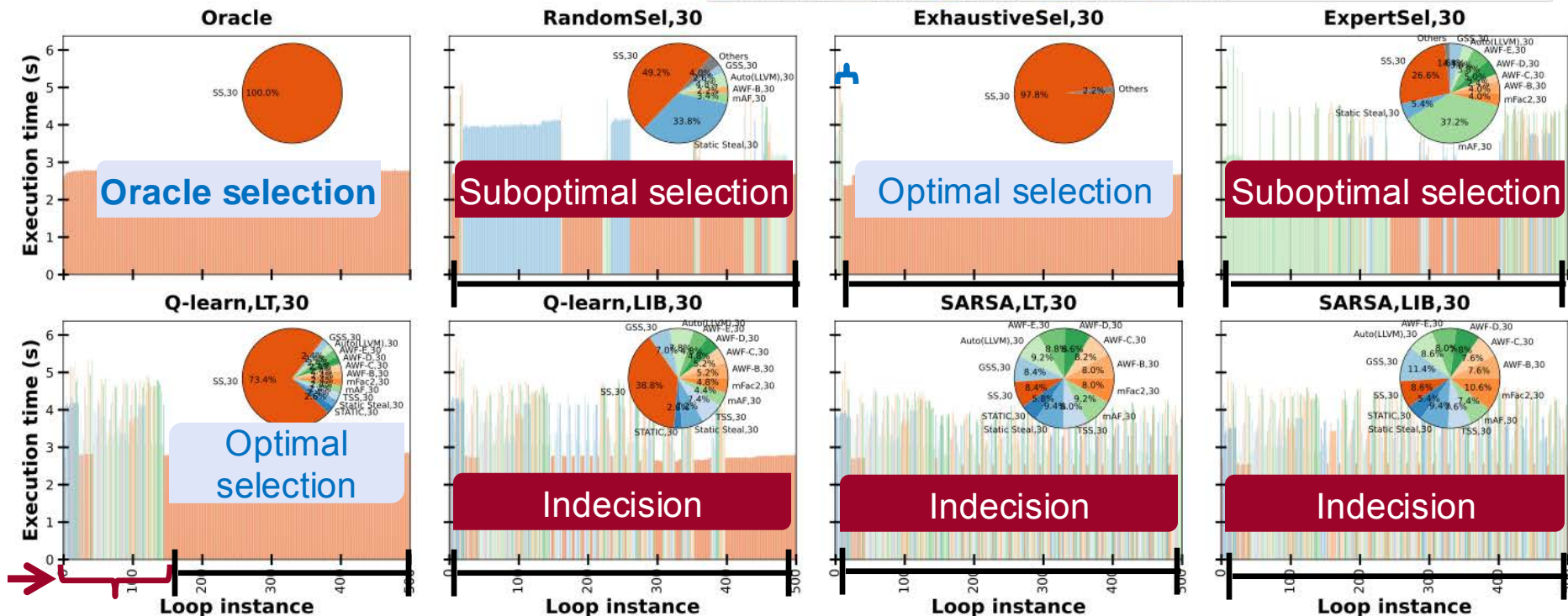


Certain algorithms cause extreme overhead (>60% compared to oracle)

Expert chunk parameters improves performance in most cases

Results: Comparison Between Expert-based vs. RL-based Selection

SPHYNX Evrard collapse | L0 | expChunk | EPYC

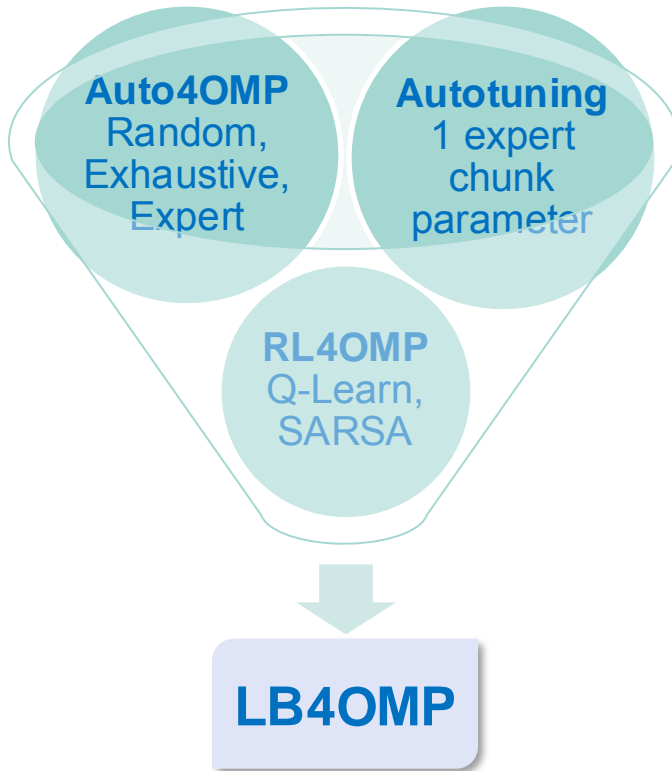


➔ Long exploration phase

➔ Short exploration phase

⏏ Exploitation phase

Take Away's



<http://github.com/unibas-dmi-hpc/LB4OMP>

Q-learn and SARSA are not ideal for scheduling algorithm selection in OpenMP

For now, **expert-based** selection **outperform** **RL-based**

Exploration of **RL-based** algorithms that require **less steps for learning** is required

LB4OMP portfolio of **scheduling algorithms** is already available in **DAPHNE** (DaphneSched)



DaphneSched



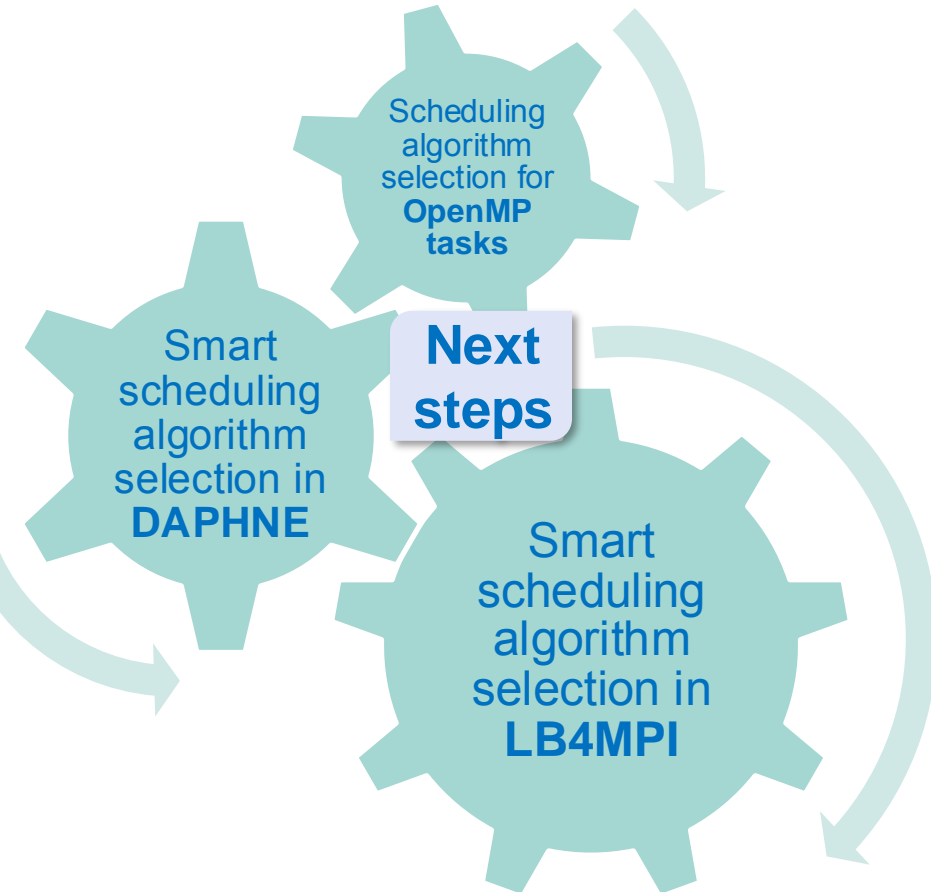
DMI HPC Group website



LB4OMP



LB4OMP w/ Auto4OMP





SC24 Booth Talk Series

openmp.org

OpenMP API specs, forum,
reference guides, and more

link.openmp.org/sc24talks

OpenMP SC24 booth talk
videos and presentations