

Who Decides?

Beyond Runtime Defaults, Toward Adaptive Multi-Device OpenMP

Florina M. Ciorba

Joint Keynote II

IWOMP & EuroMPI 2026 · TU Wien, Vienna, Austria

October 8, 2026, 9:00–10:00

florina.ciorba@unibas.ch | [@cflorina.bsky.social](https://bsky.social/@cflorina) | www.linkedin.com/in/cflorina

What We Build in Basel

github.com/unibas-dmi-hpc/LB4OMP · deepwiki.com/unibas-dmi-hpc/LB4OMP

Applications

SPH-EXA · DAPHNE pipelines · Family Gene Toolkit · proxy applications



Middleware: models, libraries, runtimes, tools

LB4OMP · Auto4OMP (Exp4OMP + RL4OMP) · LB4MPI · DLS4LB · LB4OACC · DaphneSched

System management: jobs, allocation, monitoring

Slurm · Flux · monitoring and reproducible benchmarking

System software: OS-level, file systems, storage

Hardware: cores, accelerators, memories, network

Each row below applications makes a decision on their behalf:

- which **chunk, task or device** (middleware)
- which nodes (system management)
- which core, now (system software)
- which queue (hardware).

All these are questions the programmer did not ask, and is not told the answers to.

More than 10 years of my group's work sits in **the row that decides.**

→ *Two years ago, somebody described the same row from the MPI side...*

“

The MPI runtime runs the show.

”

M. Schulz “From ‘MPI+OpenMP’ to ‘MPI in harmony with OpenMP’”

He was not bragging nor complaining.

But pointing out that MPI was designed for a machine we no longer have (single core).

IWOMP 2026 Today: Let Me Ask

what a default costs

“

Who runs the **OpenMP** show?

”

What it costs when the answer is “*Whoever shipped the default!*”?

Recall IWOMP / EuroMPI 2026 Yesterday's Joint Keynote I

A. Herten (JSC), “**From JUPITER to the Stars**”: Europe's first exascale machine.

→ **Today**: *who decides how that machine is used (for OpenMP programs)?*

Six Decisions Your OpenMP Program Already Delegates Without Negotiation

Delegation to defaults...

1 Which schedule kind

decided by
compiler and OpenMP runtime
▶ **our evidence**



2 Which chunk size

decided by
OpenMP runtime,
per loop, per dispatch
▶ **our evidence**



3 Which task runs next

decided by
the task scheduler,
at every scheduling point
▶ **others' evidence**

4 Which device

decided by
the runtime,
via the default-device rules
▶ **others + our BSc theses**



5 How many teams, of what shape

decided by
compiler and device runtime
▶ **others' evidence**

6 Which core, in which order

decided by
the operating system,
mid-region, while the parallel
region runs, without notice
▶ **our OS study**



Six decisions, all delegated, none negotiated.

→ Let's travel today through history and relevant work. First stop: Defaults.

Today's Journey

Batch scheduler*

which nodes, when, for how long

MPI runtime

which rank gets which work

OpenMP runtime

which schedule kind, which chunk, which task, which device

Operating system

which thread runs on which core, now

Hardware*

cores · accelerators · memories

* not discussed today

1

Where defaults come from

history and design intent

2

The OpenMP runtime

the loop, where we could measure ·
then tasks and devices: others and a bit of our evidence

3

The layer below

the operating system re-decides

4

The layer beside

MPI: *above* OpenMP in the stack but a **peer** in the decision
· joint scheduling

The fixed defaults were reasonable for 2008 nodes they were written for; on today's nodes, same defaults cost.

OpenMP Specification Is “Silent” on Mapping Work to Hardware

runtimes fill the gap, and users depend on it

What the specification fixes: the meaning of the program

**semantics: what each construct is
required to do**

- correctness and the memory model
- what a conforming program may rely on

What it leaves to each implementation, *deliberately*: the mapping of work onto hardware

**mapping: which iterations form a chunk, and when
chunking stops being static**

- the order in which ready tasks run
- how many teams, of what shape, on which device
- on which core each thread ends up executing

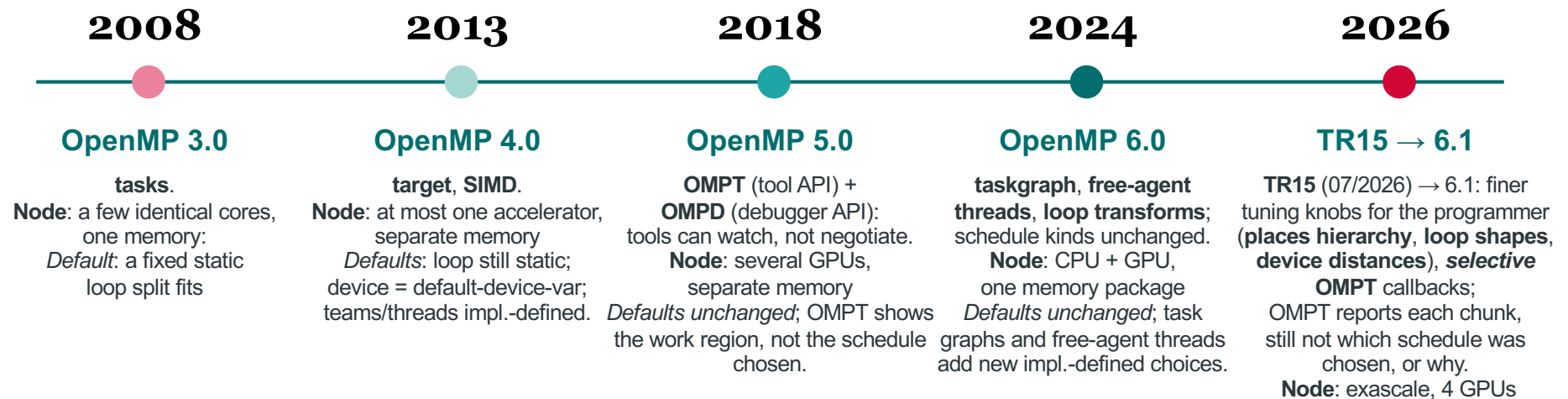
This deliberate silence keeps **OpenMP implementable** on every machine. 😊
Implementations fill it with **similar implementation-defined behaviors**
 (“quasi-standards*”), and users tune their codes to them.

→ *Why those behaviors looked right, once upon a time...*

* Tuft, Weinzierl, Klemm (IWOMP 2024, LNCS 15195) found this for tasks: the specification is “deliberately unspecific” about how a task graph runs, yet LLVM, GCC and Intel runtimes behave alike (“quasi-standards”), and codes are tuned to it. The same holds for loop schedules, devices and placement.

Fixed Scheduling Defaults Were the Right Call... At their Time

“the default” = what the OpenMP runtime does when the program says nothing: schedule kind (e.g., def-sched-var: impl. defined) and chunk, task order, device, teams



The defaults did not change. The workload, machine, & how they meet, **did.**
→ *What does an incorrect scheduling default actually cost?*

openmp.org/specifications · ARB press release, 9 Jul 2026: TR15 is the final public comment draft; 6.1 “expected shortly before SC26 in November 2026”.

What an Incorrect Scheduling Default Actually Costs

a wrong default = a fixed choice that does not fit the loop, the data, or the machine · four costs

Not only a **performance** cost, but also...

Performance

The incorrect chunking **may leave threads idle** at the barrier, every time step.

Portability

Code tuned around one default **may strand on the next device**, with the source unchanged.

Productivity

The **tuning burden lands back on the domain scientist**, who did not sign up for it.

Sustainability

Idle cores and repeated tuning runs are energy spent for nothing.

Four costs, paid together each time work meets a device.

→ *Why defaults are incorrect, more often, today?*

What Changed? (1) The Workload Stopped Holding Still

irregular · pipelines · AI in the loop · ensembles

Irregular by construction

E.g., the neighbor count per SPH particle **changes** as the simulation **evolves**.

Coupled pipelines

E.g., DAPHNE pipelines:
Simulation →
data processing →
training
in **one job** (not three).

AI now in the game

Surrogate models are steps **inside** the simulation,
not before or after it.

Ensembles, not single runs

Many short jobs; the **decision extends up** to the batch scheduler (system management) too.

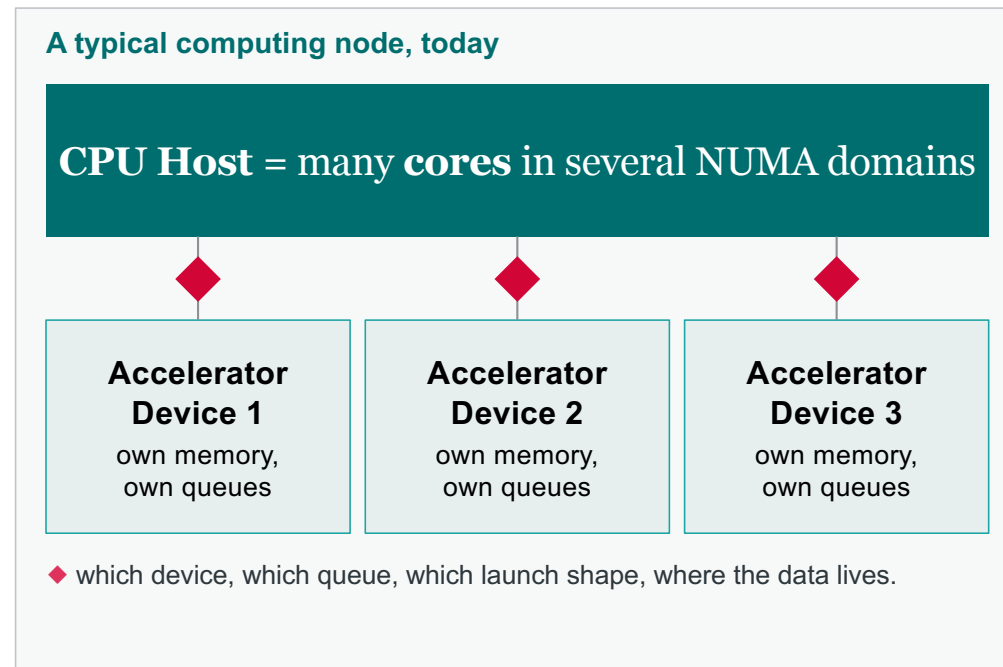
A decision made once, before the run, by whoever wrote the directive, e.g. `schedule(static,64)`, or the runtime default when programmer typed nothing, cannot follow any of this.

→ *What about changes in the node?*

SPH-EXA and DAPHNE: joint work with R. M. Cabezón, A. Cavelan, A. Eleliemy, J. H. Müller Korndörfer, Q. Guilleateau, O. S. Simsek, S. Keller.

What Changed? (2) The Node Stopped Being One Thing

several devices · different vendors · deeper memories · queues and async · ♦ = something chooses for you



♦ Several devices per node

default-device-var = which device runs a target region

♦ Several vendors per machine

each compiler picks its own teams and threads

♦ Deeper, non-uniform memories

which memory (NUMA, HMB) holds the data is part of the schedule

♦ Queues and asynchrony

runtime choice = which queue, how far the host waits for it: the whole device context, one stream, or one event

Something chooses for you per node, machine, memory, queue, synchronization.
→ *And what do Top500 nodes look like in 2026?*

What Changed? (2) The Node Stopped Being One Thing: Very True Today!

TOP500, June 2026 · very many cores per node · MPI across nodes, OpenMP across cores and accelerators

LineShine, TOP500 #1, June 2026

2.198 Eflop/s HPL, +20 % over El Capitan; also #1 on HPCG

22,680 nodes: 2 × LX2 per node = 2 × 304 Arm cores (SVE vectors + SME matrix) + HBM on package

⚠ No GPUs / discrete accelerator. No separate memory space.

JUPITER Booster, TOP500 #5, June 2026, Europe's #1

1.000 Eflop/s HPL, 55 % of El Capitan (45 % of LineShine);

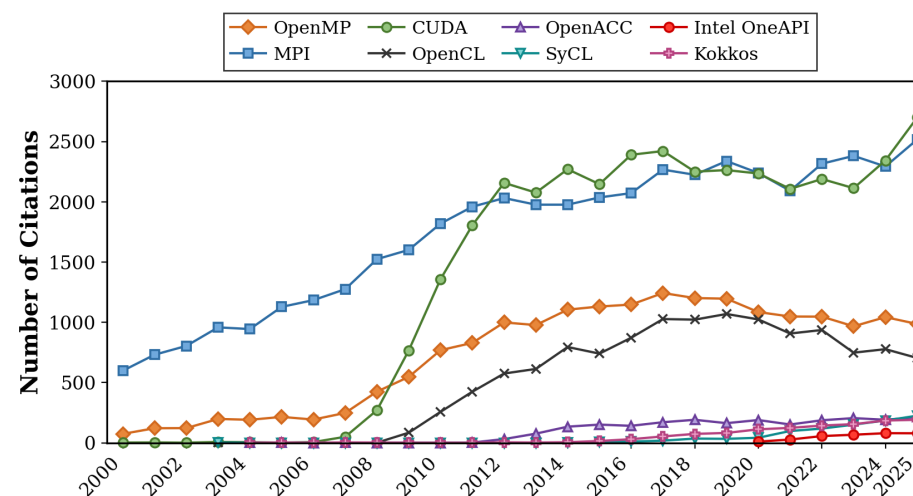
no HPCG result yet

5,884 nodes: 4 × GH200 per node = 4 × 72 Arm cores + 4 × H100

GPUs (96 GB HBM3 each) = 288 CPU cores +

⚠ 4 GPUs / node, each with its own memory, next to the host's.

Which programming model(s)?



MPI and OpenMP still dominate.

Citations in Scopus of the most used parallel programming models, 2000–2025 (non-exhaustive):

Plot: F. Ciorba, J. H. Müller Korndörfer, extended from P. O. A. Navaux, A. F. Lorenzon, M. da Silva Serpa, J. Braz. Comput. Soc. 29(1), 2023.

Two of the oldest models can run the fastest machines of 2026; 608 cores (LineShine) or 288 cores + 4 GPUs (JUPITER) per node still need a schedule! → *How does workload meet node?*

Sources: TOP500 & Green500, June 2026 (67th ed.), top500.org/lists/top500/2026/06 · LineShine: HPCwire 25 Jun 2026, ServeTheHome 23 Jun 2026, arXiv:2605.08633 · JUPITER: JSC, "JUPITER Technical Overview", fz-juelich.de/en/jsc/jupiter/tech; HPCwire, 24 Jun 2026, "JUPITER Ranked 5th on TOP500"; TOP500 news 23 Jun 2026 (HPCG: not submitted).

Who Decides? Beyond Runtime Defaults, Toward Adaptive Multi-Device OpenMP

IWOMP | EuroMPI 2026 Keynote | F. M. Ciorba

12

What Changed? (3) Workload Meets Node and Scheduling Is the Bottleneck

agentic AI = a GPU model calling CPU tools in a loop · host and device wait for each other · the published fix is a scheduler

Agentic AI is a CPU workload

Tool calls (run code, write files, call APIs), parsing, safety checks and re-tokenizing **run on the CPU**. “7 of 8 stages” [1]

Both host and device idle in turn

While the GPU runs the model, **the CPU waits**; while the CPU runs tool calls, **the GPU waits** [2]

The proposed fix is scheduling

COMB scheduler cuts end-to-end latency (queueing + service) by up to 1.8× under sustained load [2]

And core count decides

Too few CPU cores and the **GPUs stall**. More cores **cut time-to-first-token (TTFT)** 1.5-7× at long prompts [3]

What happens when a workload that won't hold still runs on a node that's not one thing?
Scheduling with fixed choices/defaults between them **breaks!**

→ *Meet adaptive scheduling decisions...*

Exemplified by (not limited to) recent studies of agentic-AI serving:

[1] M. S. Smith, “The CPU Comeback Is Upon Us”, IEEE Spectrum, 16 Aug 2026 (7 of 8 agentic AI stages, M. Rangarajan) cites [2] and [3] below.

[2] R. Raj, S. Kundu (Intel), I. Vohra, H. Wang, T. Krishna (Georgia Tech), “Towards Understanding, Analyzing, and Optimizing Agentic AI Execution: A CPU-Centric Perspective”, arXiv:2511.00739v3, Apr 2026 (COMB: **C**PU-Aware **O**verlapped **M**icro-**B**atching scheduler).

[3] E. Chung, Y. Jia, A. Jezghani, H. Kim (Georgia Tech), “Characterizing CPU-Induced Slowdowns in Multi-GPU LLM Inference”, arXiv:2603.22774, 2026 (1.5–7× TTFT range).

Adaptive Scheduling Decisions ...

“Adaptive” means the choice is made, and revised, during the run from what the runtime observes, instead of once, by a default.

... must become **first-class**, **observable**, and **cross-layer**.

The answer to “**Who decides?**” is “**The runtime**”,
asked to adapt, in the open, together with its neighboring runtimes.

First-class

The program **can ask** for **adaptive selection of schedule and chunk**, and **an objective**, and the runtime commits to it.

Observable

Which schedule and chunk were chosen, and **when they changed**, is visible to tools.
Today only the chunk is. 😊

Cross-layer (or multilevel)

The **OS below** (which core) and **MPI beside** (which rank) **decide at the same time** about the same cores.

Runtime adapts to the program and the node, not the other way around.
Everything after this slide is **evidence for this claim**, *where I have it*.
→ *The OpenMP (worksharing) loop...*

Today's Journey

Batch scheduler*

which nodes, when, for how long

MPI runtime

which rank gets which work

OpenMP runtime

which schedule **kind**, which **chunk**, which task, which device

Operating system

which thread runs on which core, now

Hardware*

cores · accelerators · memories

* not discussed today

1

Where defaults come from

history and design intent

2

The OpenMP runtime

the loop, where we could measure ·

then tasks and devices: others and a bit of our evidence

3

The layer below

the operating system re-decides

4

The layer beside

MPI: *above* OpenMP in the stack but a **peer** in the decision

· joint scheduling

Loop decisions: schedule kind and chunk, where we could measure) then tasks and devices.

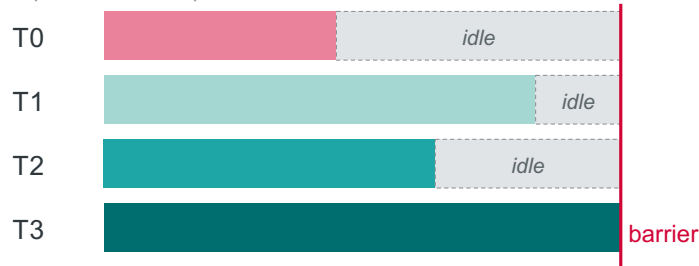
What the Runtime Is Choosing on Your Behalf, by Default

two lines of code · the default: static, equal chunks · one barrier, everyone waits · nobody tells you

```
#pragma omp parallel for
for (i = 0; i < N; i++) compute(i);
```



Equal chunks, unequal work: each thread runs its chunk, then waits at barrier for slowest (T3)



Colors = a thread computing its chunk; **gray** = idle at the **barrier**.
Chunks split is fixed before the loop starts → no idle thread can take over part of T3's chunk:
with **static** there is **no shared queue** to take from (**dynamic**, **guided** or **static_steal** would allow it, at the cost of data movement).

Code lines say nothing about chunks, so the default applies.

The OpenMP specification leaves the default **schedule** to the implementation; every mainstream runtime uses **static** (quasi-standard): equal-size chunks, one per thread, fixed before the loop starts.

⚠ **Equal iterations are not equal work:** the right iterations split depends on the loop body, the input data, the machine, and whatever else runs on the node at the same time.

Nothing in the run reports this default (other standard) choice(s), or whether it fit the loop / data / machine/...
→ *What if there were other choices?*

LB4OMP: The Literature, Made Runnable

24 algorithms · 1 binary code · chosen during execution through `OMP_SCHEDULE` · extends the LLVM OpenMP runtime

1

static

15

dynamic

7

adaptive

1

work-stealing**

24

in total

* in the OpenMP specification; ** only in the LLVM runtime, not spec

STATIC*	SS*	GSS*	Static Steal**	FSC	mFSC
TSS	TFSS	FAC	mFAC	FAC2	WF2
TAP	BOLD	FISS	VISS	RND	AWF
AWF-B	AWF-C	AWF-D	AWF-E	AF	mAF

LB4OMP extends the LLVM OpenMP runtime (2020); open source.

Durand, Broquedis, Gautier, Raffin (IWOMP 2013) added an **adaptive, NUMA-aware work-stealing loop scheduler** to GCC's libGOMP;

LLVM's **static_steal**** is its closest relative, and it is in the **LB4OMP** portfolio.

40 years of self-scheduling literature. 😊

! Point was never that *1 of these is best*.

They are all **selectable** during execution, from same binary, through `schedule(runtime)` and environment variable `OMP_SCHEDULE`.

“Which one?” becomes a measurement, not a wish or a myth.

→ *Who chooses among them?*

J. H. Müller Korndörfer, A. Eliemy, A. Mohammed, F. M. Ciorba, “LB4OMP: A Dynamic Load Balancing Library for Multithreaded Applications”, IEEE TPDS, 2022, doi:10.1109/TPDS.2021.3107775 · github.com/unibas-dmi-hpc/LB4OMP · deepwiki.com/unibas-dmi-hpc/LB4OMP
M. Durand, F. Broquedis, T. Gautier, B. Raffin, “An Efficient OpenMP Loop Scheduler for Irregular Applications on Large-Scale NUMA Machines”, IWOMP 2013, LNCS 8122

Answer 1 to “Which one?”: Expert-Based Selection

which one, for this loop, now? · 4 expert-based selection methods · a portfolio of 12 scheduling algorithms · the OpenMP runtime also picks the expert chunk

`schedule(auto)` delegates the choice of **kind** and **chunk** to “compiler/runtime system”: in effect, “*I do not know*”.

Auto4OMP takes that literally: compile the loop with `schedule(runtime)`, set `OMP_SCHEDULE="auto,<method>"`, and `<method>` becomes a **selection method** over a portfolio of **12 scheduling algorithms** (STATIC, SS, GSS, GAC, TSS, Static Steal, mFAC2, AWF-B/C/D/E, mAF). No change to the source.

RandomSel

Pick 1 algorithm at random; pick again when load imbalance > 10 %.

Baseline: how much gain comes merely from switching?

ExhaustiveSel

Try each algorithm once, keep the fastest; search again when load imbalance > 10 %.

Pays for exploring, online.

ExpertSel

Fuzzy-logic rules from the load balancing literature, fed with loop time and load imbalance, and their rates of change.

Oracle (not a selector)

The **best** algorithm per loop and time step, found **offline** by running every algorithm.

Baseline; *unaffordable in practice.*

Each schedule also sets the **chunk**: with an expert chunk, computed by the runtime from N iterations, P threads and the golden ratio $\phi \approx 1.618$ within $[1, N/(2P)]$ (TPDS 2022, Eq. 1)

ExpChunk = the size of every chunk for **dynamic** | the smallest chunk allowed for **guided**, TSS, FAC, AWF...

Choosing an algorithm **and** its configuration parameter is a more complete choice! → *Results...*

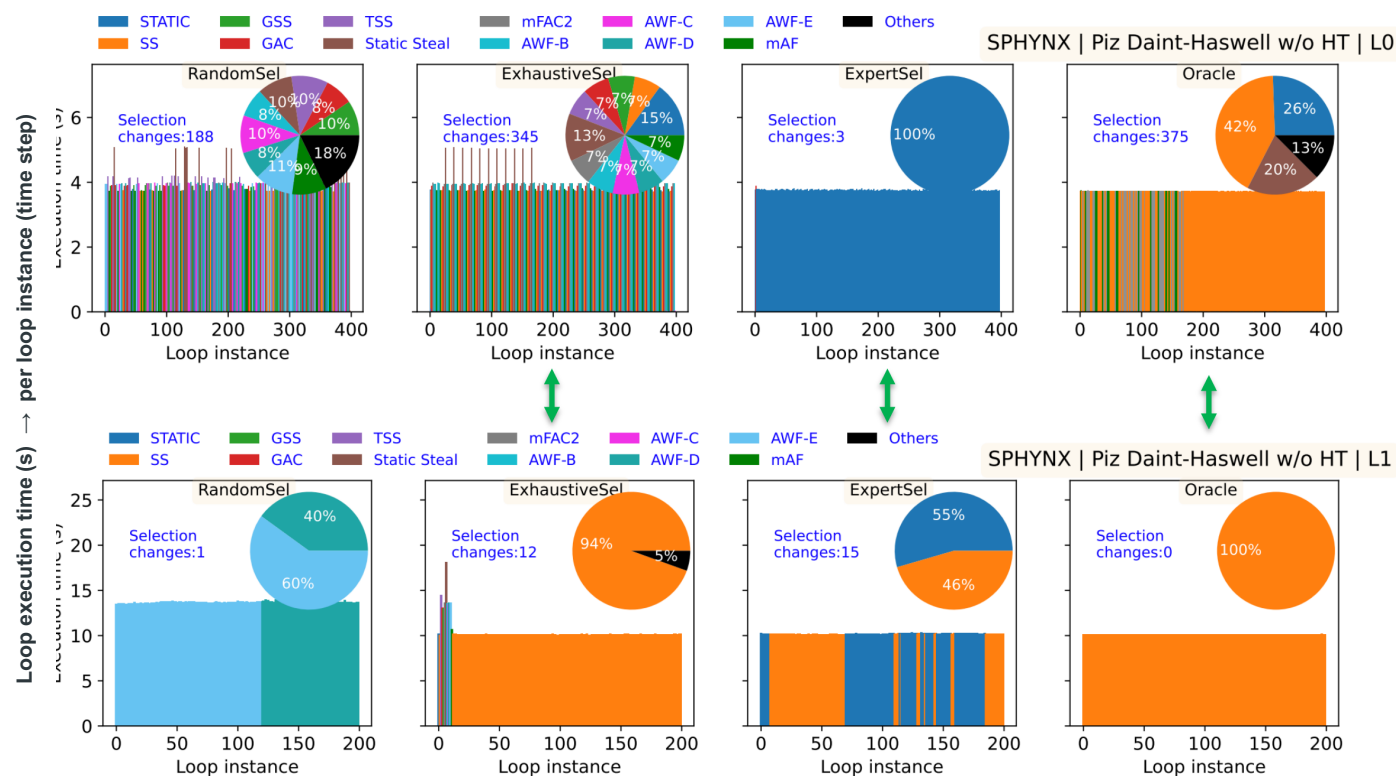
`OMP_SCHEDULE="auto,<method>"` in **LB4OMP**: method chooses among 12 algorithms

2 **RandomSel** · 3 **ExhaustiveSel** · 4 **BinarySel** · 5 **ExpertSel** (expert-based) · **Oracle** = offline baseline, not a selection method.

A. Mohammed, J. H. Müller Korndörfer, A. Eleliemy, F. M. Ciorba, “Automated Scheduling Algorithm Selection and Chunk Parameter Calculation in OpenMP”, IEEE TPDS 33(12):4383–4394, Dec 2022, doi:10.1109/TPDS.2022.3189270

Answer 1: Expert-Based Selection: Different Loops Want Different Answers

SPHYNX on Piz Daint · two heavy loops, same binary code, same cores · schedule selection changes per loop and per time step · lower is better



SPHYNX is an astrophysics SPH code

Test case: **Evrard collapse** simulation

L0 = neighbour search (findneighbours): runs 2× per time step.

L1 = gravity (tree walk): runs 1× per time step; the most expensive loop.

Piz Daint, Cray XC50: 1 × 12-core Intel Haswell per node, no hyper-threading; threads pinned (OMP_PLACES=cores, OMP_PROC_BIND=close); expert chunk on.

y = loop time per instance (x);

color = algorithm selected; pie = share of instances.

Lo and L1 do not want the same schedule (! even Oracle shows it).

Nor the same schedule twice in a row
(with exceptions).

→ And against LLVM's default itself?

Auto4OMP selection for loops L0 and L1, per time step. Source: SC22 OpenMP booth talk (H.5); loop roles: SC'20 OpenMP booth talk (LB4OMP), L. Kury, MSc thesis, Basel 2022; also Leogang 2023 s.18, VarSched 2023 s.15.

Answer 1: Expert-Based Selection Against LLVM's Default schedule(auto)

Auto4OMP vs LLVM schedule(auto) · up to 11 % · measured in 2022 · lighter is better · no change to the source

Up to 11 % slower over the Oracle.
 ⚠ measured in 2022, against the LLVM (auto) of that time.
 ~1 % Closest to the Oracle across all application-system pairs

App-Sys pair	Prior work (no or manual selection)			Auto4OMP					Oracle exec. time
	Static	Steal	GAC (LLVM-auto)	ManualBest	SS,expert chunk	RandomSel	ExhaustiveSel	ExpertSel	
ALYA-Piz Daint-Haswell	11.48%		2.44%	2.03%	2.93%	0.29%	0.23%	0.89%	11,492s
GROMACS-miniHPC-Broadwell	22.11%		0.44%	0.11%	1.42%	1.18%	0.56%	0.67%	1,954s
GROMACS-Piz Daint-Haswell	52.12%		0.27%	0.17%	5.26%	0.75%	0.49%	1.10%	22,248s
Mandelbrot-miniHPC-Broadwell		0.36%	0.38%	0.14%	0.02%	0.54%	0.36%	0.87%	1,503s
Mandelbrot-Piz Daint-Haswell		0.20%	0.37%	0.09%	0.13%	0.41%	0.37%	0.45%	8,394s
Mandelbrot-miniHPC-Cascade-Lake		0.46%	0.45%	0.35%	0.31%	1.20%	1.64%	1.42%	1,503s
SPEC OMP 2012 352.nab-miniHPC-Broadwell		4.64%	3.08%	0.20%	2.91%	2.83%	0.96%	2.42%	825s
SPEC OMP 2012 352.nab-Piz Daint-Haswell		3.84%	2.33%	1.59%	2.58%	2.30%	1.99%	2.13%	1,191s
SPHYNX-miniHPC-Broadwell		22.01%	10.36%	5.43%	0.02%	13.48%	1.58%	0.55%	3,863s
SPHYNX-Piz Daint-Haswell		38.81%	12.38%	3.04%	0.11%	15.89%	1.65%	0.50%	4,918s
SPHYNX-miniHPC-Cascade-Lake		26.08%	11.02%	2.45%	0.29%	13.71%	1.19%	1.13%	1,321s

Rows = application-system pairs: ALYA, GROMACS, Mandelbrot, SPEC OMP 2012 352.nab, SPHYNX on Piz Daint (Cray XC50, 1 × 12-core Intel Haswell per node), miniHPC Broadwell (2 × 10-core Xeon E5-2640 v4) and miniHPC Cascade Lake (2 × 28-core Xeon).

Cells = % slower than the Oracle; lighter = closer. GAC = LLVM's schedule(auto).

Table from: Mohammed et al., IEEE TPDS 2022; figure and highlight as in the SC22 OpenMP booth talk (H.1) and Leogang 2023 s.14.

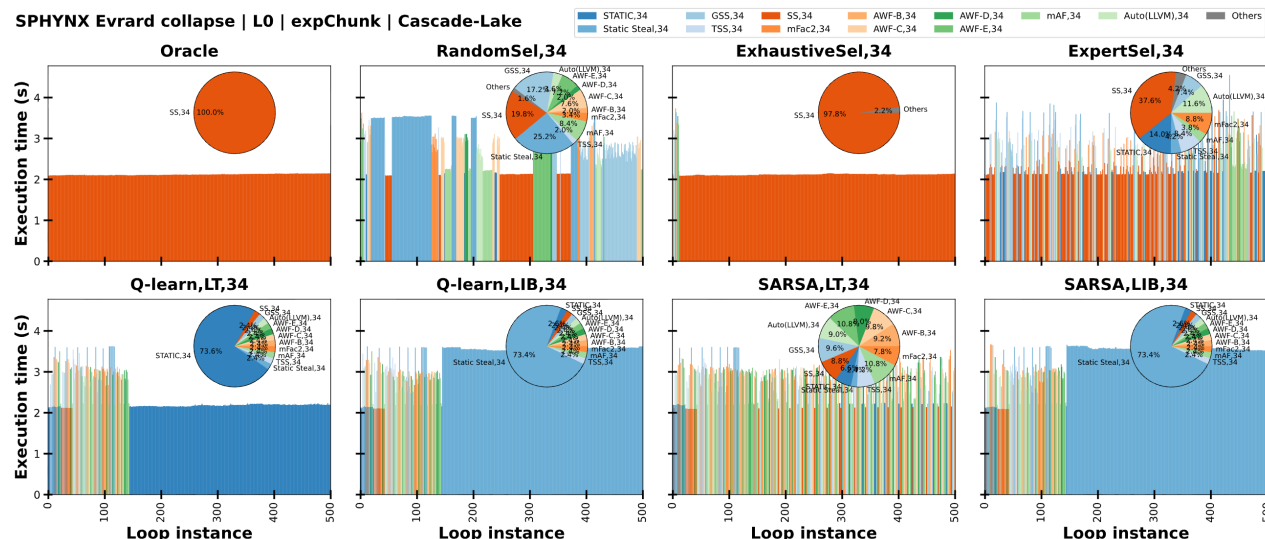
Beating the default schedule(auto) cost **no change** to the loop body:
 schedule(runtime) + OMP_SCHEDULE="auto,<method>" did it.
Can an agent learn the choice instead? → Answer 2

Why the 2022 numbers cannot be re-measured today: LB4OMP forked the LLVM OpenMP runtime in 2020; has not merged upstream since, does not build with modern Clang. Current Clang emits a call **our fork lacks** (`__kmpc_dispatch_deinit`; a stub fixes it) and no longer fills `loc->psource`, the source file & loop location string Auto4OMP uses to tell loops apart per selection; Without `loc->psource`, the adaptive runtime can still choose, but no longer knows which loop it is choosing for → Observability gap by S. J. Newcome.

Answer 2: Learning-Based Selection: RL Finds a Good Choice... Slowly



RL4OMP inside OpenMP · SPHYNX, gravity loop (L1), 500 time steps · learning costs 144 of 500 instances · the reward decides where it settles



SPHYNX Evrard collapse, gravity loop (L1), expert chunk (,34), 2 × 28-core Intel Cascade Lake.

Top: Oracle and expert-based methods; bottom: RL-based selection methods and rewards.

x = loop instance (time step); y = execution time of that instance; color = algorithm chosen; pie = share of instances.

Source: Müller Korndörfer et al., IEEE Access 13, 2025 (and SIAM PP26 talk).

RL finds a good choice by itself; pays for learning every run: 144 / 500 loop instances.

→ *Expert or learned: which is better?*

OMP_SCHEDULE="auto,<method>" in LB4OMP: method=6–14 RL agents (learn to select which scheduling algorithm) · 15 RL direct-chunk learner (not shown here, learns the chunk size of dynamic) · **Oracle** = offline baseline, not a selection method. Each chooses among 12 algorithms.

Learning costs time:

Explore-first visits **once** every (state, action) pair → $12 \times 12 = 144$ of 500 instances (28.8 %): the jagged start of each RL curve.
state = algorithm in use
action = algorithm to switch to

Loop-time (LT) reward:

Q-Learn settles on **STATIC, 34**

Oracle picks SS, 34, 2.4 % apart, within noise.

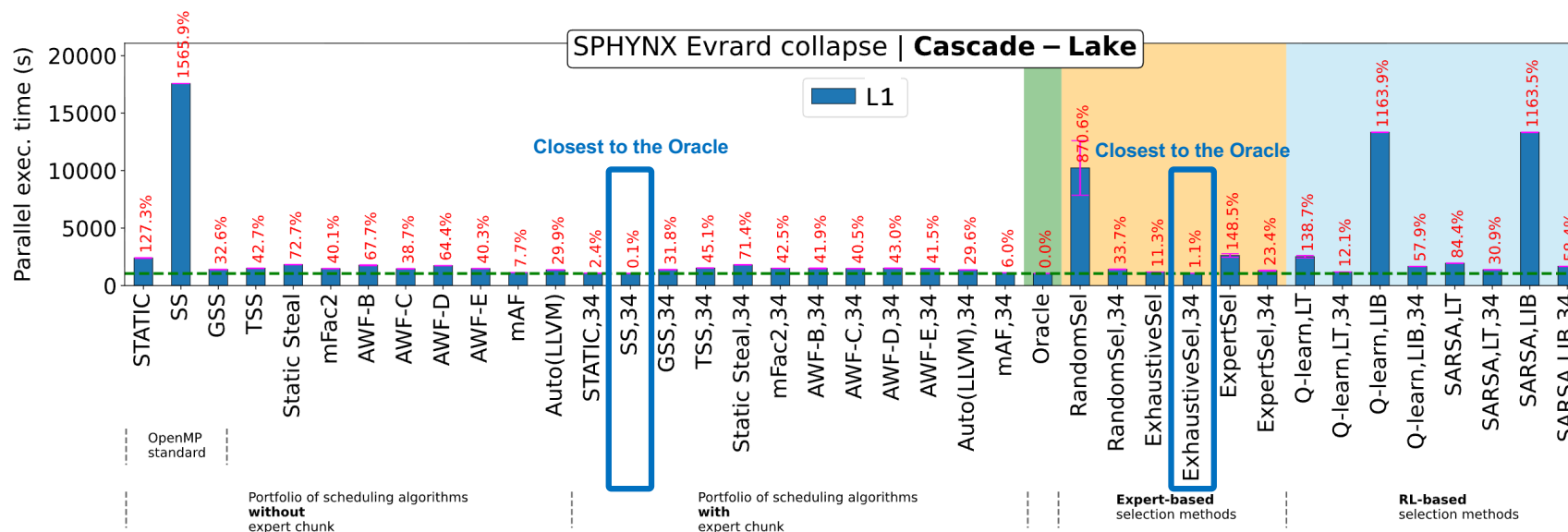
Load-imbalance (LIB) reward:

SARSA settles on **Static_Steal**, at ≈ 3.5 s per instance vs. ≈ 2.1 s for Oracle.

SARSA, LT and **ExpertSel** never settle:
gravity loop (L1) keeps changing.

Expert-Based or Learning-Based Selection? Neither Wins Everywhere

same loop, same machine, % above Oracle · STATIC default +127 % · LLVM auto +30 % · ExhaustiveSel + expert chunk +1.1 % · lower is better



SPHYNX Evrad collapse, gravity loop (Access 2025's L0), 500 time steps, 2 × 28-core Intel Cascade Lake.

Bar = parallel loop time (s); Red = % above Oracle. Green = Oracle, Orange = expert-based, Blue = RL-based; ",34" = expert chunk.

Source: Müller Korndörfer et al., IEEE Access 13, 2025, Fig. 6.

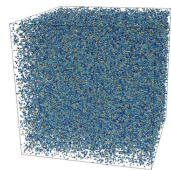
Across 6 applications × 3 systems: ExhaustiveSel+ExpChunk came closest to Oracle in every case (within up to 35 %); here SS,34 does it (0.1%).

RL adapts to new systems but needs to pay for learning; the reward type matters most. ExpChunk made RL 25-40 % better on mem-bound loops.

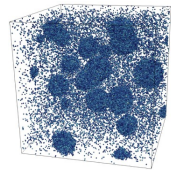
No single method wins everywhere. Right choice moved across loops, instances / loop, and across selection methods, in our codes. Does it also move in time, in somebody else's code? → *Independent evidence...*

Independent Evidence: Right Schedule Changes w/ Physics, Time Step by Time Step

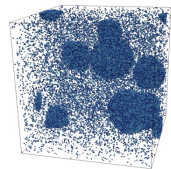
S. J. Newcome, TUM · AutoPas molecular dynamics library · one algorithm, three OpenMP standard schedules · the winner flips when the physics changes · lower is better



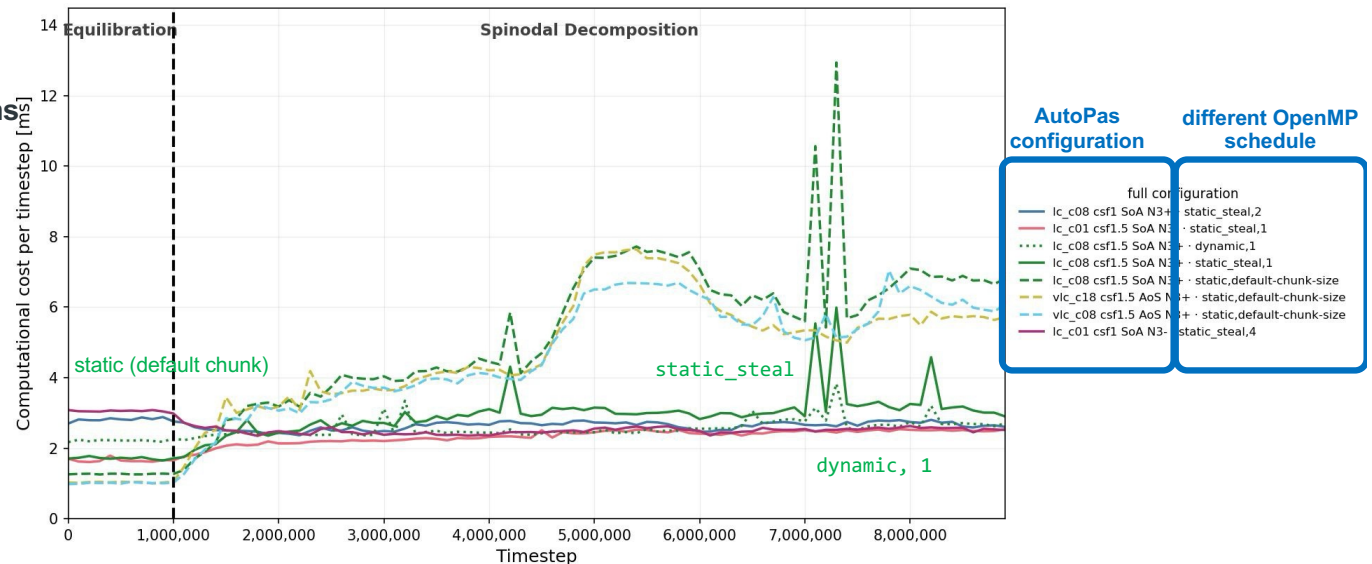
gas, equilibration
homogeneous
static, default chunk wins



spinodal decomposition
droplets nucleate
imbalance appears



liquid-vapor equilibrium
dynamic / static_steal
ahead, up to ~2.5×



Time-to-solution (y) per time step (x), density 0.17, same run; **dashed** = static (default chunk), **solid** = static_steal, **dotted** = dynamic, 1. Compare the **three green lines**: identical AutoPas algorithm (lc_c08 csf1.5 SoA N3+), **only the OpenMP schedule differs**.

Source: S. J. Newcome, Basel project summary, 8 Sep 2026, s.31 (three-density video: s.28). Snapshots: density 0.17 at three instants.

Same code + node + algorithm: static wins during equilibration; dynamic or static_steal win once droplets form, up to 2.5× apart, and **runtime cannot notice**. → *Lessons learned...*

Lessons Learned About “Who Decides” At the Loop Level

one size does not fit all · the value of adaptivity is not uniform · a default is not neutral

LESSON 1

One size does not fit all.

Evidence: the best algorithm differs across applications and machines; *which we knew!*
across loops inside one application; and across time steps inside one loop.

The value of adaptivity is not uniform. Nothing tells you in advance which case you're in.

LESSON 2

A default is not neutral. It is an unexamined bet, placed once.

Evidence: LLVM `schedule(auto)` ~11 % off the Oracle (on SPHYNX).
static default up to 2.5× slower once droplets form (on AutoPas).

A default is a decision too: made by the implementation, once, for everyone. Cost is real.

→ *What our loop work does not do (yet).*

What Our Loop Work Does Not Do (Yet)

no devices · no tasks · the overhead is real · learning needs episodes · nobody is told

No devices

Everything you just saw is **host device and worksharing loops**.
Not one offloaded kernel.

No tasks

The task scheduler makes the **same kind of decisions** and we did not touch it.

The overhead is real

Selection costs something on every re-trigger, and **exploration costs more**.

Learning needs time

RL needs enough time steps to pay for itself. Short runs do not get the benefit.

Even when the runtime chooses well, it tells nobody what it chose: OMPT has no event that names the schedule (only chunk boundaries are visible), `omp_get_schedule` returns what was requested (“**auto**”), not what was applied, and nothing is logged: Not the tool, not the program, not the OS, not MPI.

An adaptive runtime that reports nothing makes a **private guess**: it **may be right**, but no tool, program, OS or MPI can **check** it, **learn** from it, or **act** on it.

Where else can we decide? → *Tasks and Devices*

Today's Journey

Batch scheduler*

which nodes, when, for how long

MPI runtime

which rank gets which work

OpenMP runtime

which schedule kind, which chunk, which **task**, which **device**

Operating system

which thread runs on which core, now

Hardware*

cores · accelerators · memories

* not discussed today

1

Where defaults come from

history and design intent

2

The OpenMP runtime

the loop, where we could measure ·

then tasks and devices: others and a bit of our evidence

3

The layer below

the operating system re-decides

4

The layer beside

MPI: *above* OpenMP in the stack but a **peer** in the decision

· joint scheduling

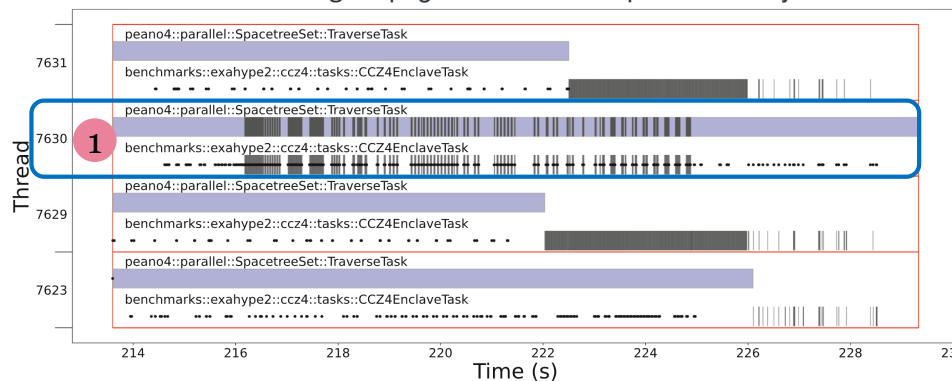
*The same delegated-and-forgotten decision,
at the task and the device.*

Tasks: Same Problem, Found Independently. Opposite Remedies.

four costly behaviors of the LLVM task runtime (Tuft, Weinzierl, Klemm, IWOMP 2024) · their remedy: programmer control · mine: adaptive and reported

Detrimental Task Execution Patterns

- 1 Tasks start too early**
queue > 256 tasks → new tasks run **at once**, not deferred
- 2 No parallelism inside tasks**
a `#parallel for` inside a task gets no extra threads
- 3 taskyield is not fair**
only a hint: cannot drain the queue, so tasks starve
- 4 Waits favor throughput**
`taskwait / taskgroup` ignore the critical path's latency



Tuft et al.'s remedy: take the decision back

- give programmer **prescriptive control** over task graph mapping
- **new clauses**: must-defer tasks, taskyield latency | throughput
- **cost**: re-supplied per machine, by hand

My & others remedy: leave it, make it accountable

- programmer cannot know the input or the node → **OpenMP runtime decides**, adaptively (not once), reports what it chose & when it changed.
- ∃ work for **adaptive loops on a work-stealing runtime**: Durand et al. [2] (libGOMP), **work assisting**: de Wolff & Keller [3].

Other trace (OMPT-based task tracer) of ExaGRyPE, one time step, 4 threads:

Bars = traversal tasks, **dots** = enclave tasks being spawned, **dark stripes** = enclave tasks executing, late & bunched after traversal.

Bars embedded in traversal task on **thread 7630 (on the critical path)** show the **task being suspended** when the thread immediately executes an enclave task. Not deferring enclave tasks prolongs the critical path.

Same problem as at loop, opposite remedies! Runtime should keep the decision, make it known.

More recent: IWOMP / EuroMPI 2026 — yesterday 15:30, Session III: Kraus, Jenke, Terboven, "Efficient OpenMP Target Task Scheduling"; Kasmeridis, Dimakopoulos, "OpenMP Tasking on the GPU"

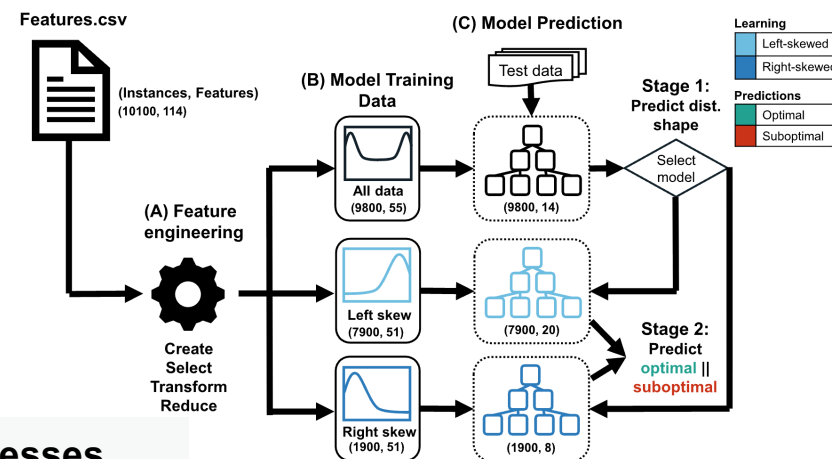
[1] A. S. Tuft, T. Weinzierl, M. Klemm, "Detrimental Task Execution Patterns in Mainstream OpenMP Runtimes", IWOMP 2024, LNCS 15195, pp. 210–224, doi:10.1007/978-3-031-72567-8_14 · [2] Durand et al., IWOMP 2013, LNCS 8122 · [3] de Wolff, Keller, ARRAY '24

Devices: Somebody Is Already Automating This

“OpenMP provides default offloading configurations” · a model ranks launch configurations · Andrews, Yi, Lin, Doerfert, IWOMP 2025

“ Unlike CUDA support, OpenMP provides default offloading configurations.

A. Andrews, Q. Yi, P.-H. Lin, J. Doerfert, “Predicting Performance for OpenMP GPU Parameter Choices”, IWOMP 2025.
In CUDA the programmer must choose grid and block sizes;
in OpenMP the implementation chooses, by default, number of teams (GPU blocks) and threads per team



A model can rank the guesses

a two-stage surrogate classifier for launch configurations: **optimal** or **suboptimal**?
 $\approx 91\%$ accuracy on unseen instances,
 $\approx 70\%$ on unseen benchmark sets;
 Not better parameters than default, but cuts the sweep of finding out: $O(n^2) \rightarrow O(\log n) - O(n)$

The default is a guess

team count and thread count are chosen by the implementation

Same question as our loop work

same layer (OpenMP runtime),
different group, different method

The device default is a guess too, and a model can rank the guesses. → Our look at devices.

IWOMP 2025, LNCS 16123, pp. 133–148, Springer 2026, doi:10.1007/978-3-032-06343-4_9 · trained on HeCBench; data set and workflow on GitHub. Doerfert is a co-author.

Devices: Same Source, Same GPU, Three Different Deciders

SU3_Bench on one A100 · GCC OpenMP target offloading · GCC OpenACC · NVHPC OpenACC · two orders of magnitude in launch geometry · three scopes of synchronization

SU3_Bench (lattice-QCD proxy; 48⁴ lattice, 12'000 it. + 2 warm-ups), 1× NVIDIA A100 (40 GB), 1 node, 1 Slurm script, same source:

Offloading · Compiler	GPU threads / kernel	Block (x·y·z)	Host-side completion primitive (12'002 kernel launches each)
OpenMP target · GCC 13.3	82'944	(32, 8, 1) = 256	cuCtxSynchronize: context-wide, after every kernel (+ allocations, frees, copies 40 B)
OpenACC · GCC 13.3	221'184	(32, 1, 1) = 32	cuStreamSynchronize: stream-wide, 1 / kernel
OpenACC · NVHPC 20.11	8'388'480	(128, 1, 1) = 128	cuEventSynchronize: event-scoped, 37'424 calls ≈ 3 / kernel

Threads per launch = how finely the loop is cut over the GPU (the **device's chunking**, decision 2);

Block = how those threads are grouped per SM, x·y·z threads per block (the **team shape**, decision 5);

Completion primitive = how far the host waits (the **barrier's scope**).

All three are decided by **compiler + device runtime**; *none appears in the source*.

Kernel launch geometry **differs** by 2 orders of magnitude (82,944 vs 8,388,480 threads/launch), with 3 synchronization scopes; the user **may** see these directives as **equivalent**. → Chosen by compiler and its device runtime, never by source.

Should the OpenMP runtime be allowed to change its mind mid-run?

→ *And who is told when it does?*

See also at IWOMP / EuroMPI 2026 Yesterday 10:30: A. Kallai, S. Pophale, S. Lee, S. Chandrasekaran, "OpenMP V&V Suite: Status of OpenMP Compilers post ECP"
W. Elwasif, D. E. Bernholdt, S. Pophale, "HeCBench-Fortran-OMP" (Fortran sibling of the HeCBench suite Andrews et al. trained on).

R. Gondal, "Investigating OpenACC Offloading", and A. Jasari, "Investigating OpenMP Target Offloading", BSc theses, HPC Group, University of Basel, Feb 2026; supervisor Dr. O. S. Simsek, advisor Prof. Dr. Florina Ciorba
Counts of launch configurations and synchronization calls only; no performance comparison (cluster NVHPC toolchain from 2020).

Today's Journey

Batch scheduler*

which nodes, when, for how long

MPI runtime

which rank gets which work

OpenMP runtime

which schedule kind, which chunk, which task, which device

Operating system

which thread runs on which core, now

Hardware*

cores · accelerators · memories

* not discussed today

1

Where defaults come from

history and design intent

2

The OpenMP runtime

the loop, where we could measure ·
then tasks and devices: others and a bit of our evidence

3

The layer below

the operating system re-decides 🤔

4

The layer beside

MPI: *above* OpenMP in the stack but a **peer** in the decision
· joint scheduling

The OS makes the same decision about the same cores, and was never told the OpenMP code exists.

Two Schedulers, Two Objectives, Same Physical Cores

cooperative vs. competitive · finish the region vs. thread fairness · no channel between them

The OpenMP runtime sched... is cooperative

- a team of threads work toward one barrier
- **objective**: finish this parallel region quickly
- **assumes** it owns the cores it was given
- progress measured in **chunks** and **time steps**

The operating system scheduler... is competitive

- every runnable thread is an equal claimant for cores
- **objective**: fairness & responsiveness (for user apps)
- **knows nothing** about the (OpenMP) barrier ahead
- progress measured in **time slices**

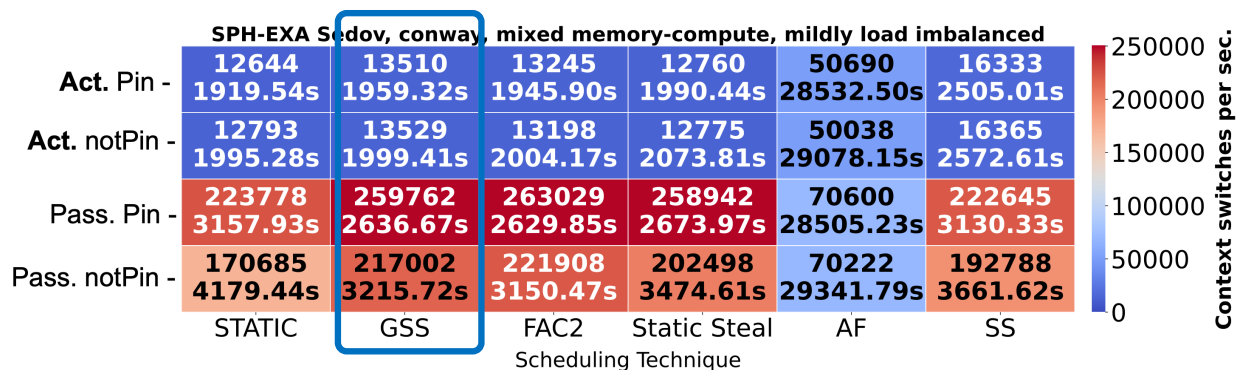
Neither is wrong. But there is no exchange channel between them.

→ *What happens when these schedulers meet?* 🤔

J. H. Müller Korndörfer, A. Eleliemy, O. S. Simsek, T. Ilsche, R. Schöne, F. M. Ciorba, "How Do OS and Application Schedulers Interact? An Investigation with Multithreaded Applications", Euro-Par 2023, LNCS 14100, pp. 214–228, doi:10.1007/978-3-031-39698-4_15.

The Operating System Re-Decides, and This Costs

unpinned threads · context switches and migrations · the switch is cheap, the idle time it causes is not



GSS column: C = context switches; D = the de-scheduling (idle) events they trigger; $r(\cdot)$ = rate in events/s; $f(\cdot)$ = share of loop time T_c they cause.
 $r(C) \approx 260 \text{ k ctxt/s}$ costs only $f(C) \approx 0.4\%$ (one switch takes μs). A D event needs a pair of switches (off, back on), so $r(D) \leq r(C)/2$, but each one idles a thread until the kernel re-replaces it and the whole team waits at the barrier: $f(D) \approx 40\%$.



Influence of non-application events on SPH-EXA Sedov blast on Conway (2×32 -core AMD EPYC 7502, Linux 6.0.3)
 Rows = OpenMP wait policy (Active: waiting threads spin on their core; Passive: threads sleep, OS kernel must wake and replace them) $\times$ pinned (OMP_PROC_BIND / OMP_PLACES: the OMP runtime asks the OS not to migrate) or notPinned.
 Columns = scheduling technique. Cell = context switches/s (top), loop execution time T_c (bottom).
 Source: Müller Korndörfer et al., Euro-Par 2023 talk, s.32 and s.34.

Not pinned threads:

OS scheduling events interfere measurably via context switches and migrations. *We knew this.*

Context switch (save state, reload, cold caches) takes $\mu\text{sec} \rightarrow$ at 260k context switches/s $f(C) \approx 0.4\%$ of T_c .

Each context switch takes a thread off its core; under the passive policy it sleeps until the kernel re-replaces it, and the other threads wait for it at the next barrier: that idle time is $f(D) \approx 40\%$ of T_c .

OpenMP runtime cannot see why its threads got slower.

“Just pin the threads” is a fair objection: pinning removes most migrations (Persson et al., ROSS’25). But it is a **static decision, placement fixed once**, before the run, for every phase, and a pinned thread can still be preempted and left idle.

An adaptive runtime blind to the OS will faithfully adapt to noise. The runtime cannot see the OS.

Can a performance tool see the runtime (which schedule kind, chunk it applied to each loop, and when it switched)? Question for OMPT!

C. Persson, M. Prétot, M. Cui, M. Pericàs, “Reproducible Performance Evaluation of OpenMP and SYCL Workloads under Noise Injection”, ROSS’25 @ SC’25, doi:10.1145/3731599.3767538.
 Bristot de Oliveira et al., IEEE TC 72(1), 2023 (osnoise, timerlat).

Can a Tool at Least See What the OpenMP Runtime Decided?

OMPT = OpenMP's standard tool interface · TR15 (6.1 draft): the chunk yes, the algorithm no, why nothing, when not reported

What was chosen?	which algorithm, which chunk , for which loop, handed out at which dispatch (each time a thread fetches its next chunk of a worksharing loop)	TR15: the chunk, yes. The algorithm, no.	a dispatch callback fires for every chunk, with its start and size (§40.14, §41.4.2); the work callback's work_type knows only loop/static/dynamic/guided/other (Table 19.1) → so the chunk is visible , the algorithm is not
Why?	which observation triggered the change: imbalance , time , an explicit request	TR15: nothing.	no callback, type or entry point carries a reason for a scheduling decision
When it changed?	the time step (so to attribute a regression)	TR15: not reported.	a switch to another algorithm is invisible : work_type stays "other" before and after it
What it cost?	the overhead of deciding , separated from the cost of the work	TR15: one coarse state.	one coarse state , ompt_state_overhead (§40.31): time spent deciding is lumped with all other runtime time

Every LB4OMP algorithm reports as **ompt_work_loop_other**, which is "*implementation defined*." A tool learns that something non-standard was chosen, but never what exactly. Neither work nor dispatch are on the must-support list (Table 39.2). → *Next: MPI*

According to **OpenMP Technical Report 15**, v6.1 Public Comment Draft, 9 Jul 2026 (eds. B. R. de Supinski, M. Klemm) — Table 19.1 p.455 · §40.14 p.779 · §41.4.2 pp.815–816 · §40.31 pp.793–795 · Table 39.2 p.764 · App. B.2 pp.963–965.

Today's Journey

Batch scheduler*

which nodes, when, for how long

MPI runtime

which rank gets which work

OpenMP runtime

which schedule kind, which chunk, which task, which device

Operating system

which thread runs on which core, now

Hardware*

cores · accelerators · memories

* not discussed today

1

Where defaults come from

history and design intent

2

The OpenMP runtime

the loop, where we could measure ·
then tasks and devices: others and a bit of our evidence

3

The layer below

the operating system re-decides

4

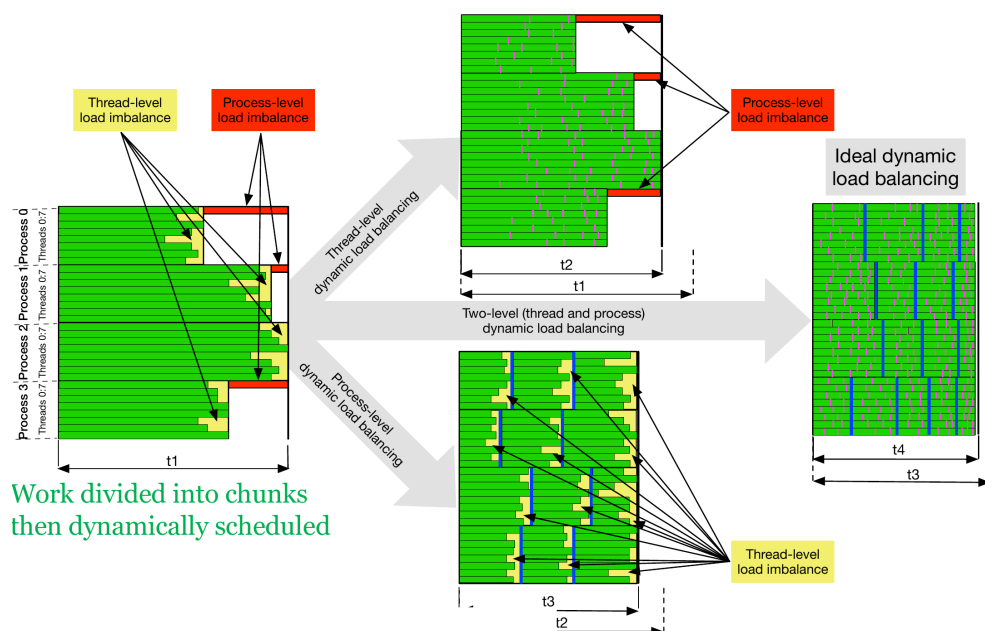
The layer beside

MPI: *above* OpenMP in the stack but a **peer** in the decision
· joint scheduling

*MPI is not below OpenMP or above it. It is **next to it**.*

Load Imbalance Lives at Two Levels (Thread, Process) & They Interact

two types of parallelism: threads inside a process, processes across nodes · two runtimes side by side



Thread- (middle, bottom), and process-level (middle, top) imbalance, their combination (left), and ideal (right).
Source: Mohammed, Cavelan, Ciorba, Cabezón, Banicescu, SIAM PP20, s.9–10

Thread imbalance

chunks of a multithreaded **OpenMP** loop **inside one process** finish at different times

Process imbalance

MPI ranks reach the communication point at different times

They interact

balancing OpenMP threads **changes** when the MPI rank arrives and therefore what the MPI level should have done; and vice versa.

Fix one form and you move the other:
load imbalance is multiruntime.

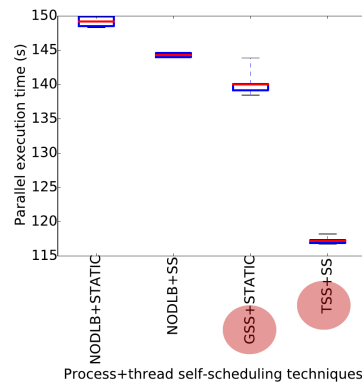
Two runtimes that could fix it sit side by side.

→ *Which runtime wins?*

Balance Load at Both, and the Best Algorithm at Each Runtime Changes

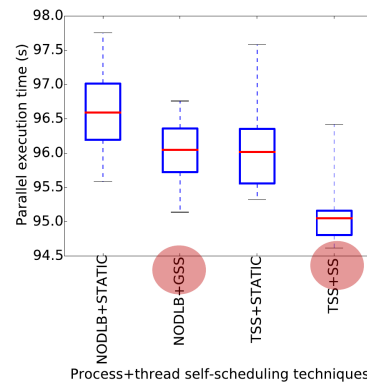


best per level, alone · best jointly · local optimality does not compose



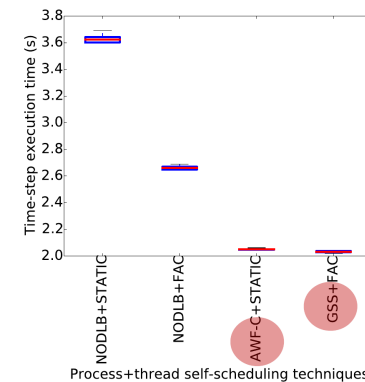
Mandelbrot

Mathematical kernel



PSIA

Computer vision



SPHYNX

Astrophysics & Cosmology

Best MPI algorithm alone.

Best OpenMP algorithm alone.

Together, neither is right any more.

Local optimality does not compose across runtimes.

Box plots of parallel execution time (y) over all MPI-level × OpenMP-level scheduling pairs (x):

Pink circles = the winner, single-level vs. two-level.

Applications: Mandelbrot (mathematical kernel, 0.6 M tasks), PSIA (computer vision, 0.8 M), SPHYNX (astrophysics, 1 M & 10 M particles).

Thread level: STATIC, SS, GSS, TSS, FAC, RAND.

Process level: NODLB, mFSC, GSS, TSS, FAC, AWF, AWF-B/C/D/E, AF.

Systems: miniHPC (20 nodes × 2 × 10-core Intel Broadwell, Omni-Path) and Piz Daint (Cray XC50, 12-core Haswell).

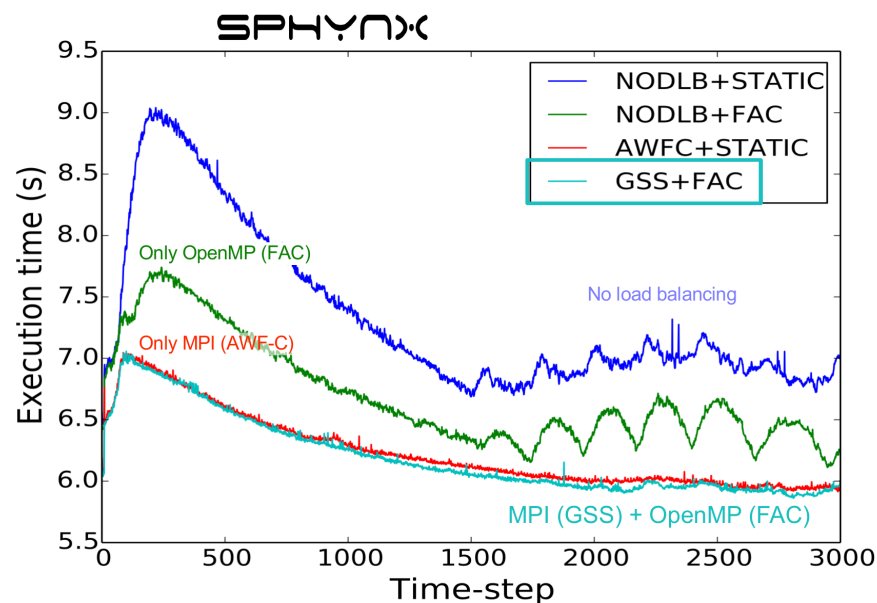
Source: Mohammed, Cavelan, Ciorba, Cabezón, Banicescu, SIAM PP20, s.17 and s.12.

Two runtimes each optimizing correctly in isolation, jointly do not get it right, **here**.

→ *Is the joint scheduling choice ever worth it?*

Yet Joint MPI + OpenMP Scheduling Is Worth It

SPHYNX, 1 M particles, 12 ranks $\times$ 10 threads $\cdot$ MPI(GSS) + OpenMP(FAC) $\cdot$ 43 % $\cdot$ 2020 hardware



43 %

MPI (GSS) + OpenMP (FAC) over *no load balancing* at either level.
OpenMP only (FAC) $\cdot$ MPI only (AWF-C)

SPHYNX Evrard collapse, execution time per time step:

1 M particles, 3'000 time steps,
12 MPI ranks $\times$ 10 OpenMP threads on 6 miniHPC nodes
(2 $\times$ 10-core Intel Broadwell E5-2640 v4).
! 2020 measurement on 2020 hardware.

⚠ Data was replicated. So, nothing had to move.

Either runtime alone leaves most of the gain on the table; no standard makes the joint choice anyone's job. $\rightarrow$ *Harmony, or adaptivity?*

A. Mohammed, A. Cavelan, F. M. Ciorba, R. M. Cabezón, I. Banicescu, "Two-level Dynamic Load Balancing for High Performance Scientific Applications", Proc. SIAM PP20, pp. 69–80, 2020. Preprint arXiv:1911.06714.

Who Decides? Beyond Runtime Defaults, Toward Adaptive Multi-Device OpenMP

IWOMP | EuroMPI 2026 Keynote | F. M. Ciorba

37

MPI & OpenMP Runtimes: Harmony, or Adaptivity?

harmonize the interfaces (Schulz) · make the decision adaptive (this talk) · two runtimes agreeing are still two runtimes guessing

Harmonize the interfaces

M. Schulz, IWOMP 2024

- let the two standards describe the same node consistently
- let **each runtime know** the other exists 🤔
- **stop** the assumptions from **colliding** ⚠️
- **his verdict**: side-by-side is “*long-term infeasible*”

Make the decision adaptive

this talk, IWOMP 2026

- an agreed interface **still needs somebody to decide**
- the right joint choice **depends** on the run
- **decided during the run**, and *reported*
- harmony without adaptivity: an agreed interface around a fixed guess

Two runtimes agreeing with each other are still two runtimes guessing.
Adaptation is key. → Harmony then adaptivity!

MPI (and M. Schulz) already asked for this (2024): more information exchange, access to each other's progress and state. **Observability first**, negotiation second.

→ **One question I cannot answer alone...**

M. Schulz, “From ‘MPI+OpenMP’ to ‘MPI in harmony with OpenMP’”, keynote, IWOMP 2024, Perth (quoted with permission)

One Question I Still Don't Know How to Answer... Alone

1 OpenMP community · similar fixes to defaults

“ If different groups (users and builders of OpenMP) independently built the same selector* for choices *outside* the specification, is then the gap in the specification? ”

* **Loop schedules** (Basel) · **Task execution patterns** (Durham, AMD) · **Offload configurations** (UNM, LLNL) · **AlgoSel AutoPas** (TUM).

1 community: those who use* OpenMP (application & library developers), those who build it (compilers, runtimes, the specification).

Gap: Spec lets implementation decide (schedule, task order, launch shape) but provides no way for adaptivity, or to learn what was decided.

→ **Fix** may consist of two halves: the research and the standard.

Five Research Directions Toward Adaptive Scheduling & Call for Funding

five research directions · and a call to funders and industry

Multicriteria scheduling

Energy, **data** movement, and resource **variability** also belong in the objective.

Learned selection

Expert knowledge and **RL** are both **algorithm selection methods**; combining RL + expert chunk cuts the learning cost.

Self-adaptive runtimes

Deciding online, revising, and being **accountable** for the revision.

Cross-layer negotiation

A **minimal protocol** by which two schedulers state what they intend to do with a core.

Fair evaluation

Reproducible benchmarking; the date and version of every measured software stack stated; variance reported.

Call to funders and industry: fund university research on adaptive scheduling.
It pays off **beyond OpenMP**: MPI, SYCL, Kokkos, task runtimes, AI serving...

Three Questions (Not Proposals) related to “Who Decides?” for the Standard

who sets the goal? · who is told? · who else decides?

1

Can a program ask for adaptivity, and state the objective? ← who sets the goal?

Today `schedule(auto)` says “*I do not know.*” It could say “*I decided this, and I optimize for that.*”

2

Can a runtime report what it chose, and when it changed? ← who is told?

Give `ompt_work_loop_other` a payload naming the schedule actually applied; put work and dispatch on the must-support list (Table 39.2).

S. J. Newcome could use, e.g., Auto4OMP to identify the loop it schedules through `loc->psource`, a debug string the compiler used to fill in; current Clang leaves it empty. Nothing was violated; nothing promised it.

Result: our adaptive runtime still chooses, but can no longer say for which loop. A reporting interface would make loop identity part of the contract.

3

Can two runtimes agree who owns a (the same) core? ← who else decides?

The OS below and MPI beside already decide about the same cores. **Nothing supports the three say so to one another.**

A channel exists on the MPI side: PMIx, the process-management interface that resource managers and MPI runtimes already speak (MPI Sessions use it for dynamic resources).

OpenMP has no counterpart; it could join that channel (or something similar) rather than invent one.

As TR15 public comments closed on 30 September; these are questions asked now for post v 6.1: better asked early and roughly than late and polished.

→ *Back to today’s question...*

I asked: “Who Runs the OpenMP Show”?

not the runtime alone · informed · observable · in negotiation with the layers above, below, and beside

The Answer: Not the runtime alone.

Better Answer: An informed, observable runtime,
in negotiation with layers above, below, and beside it.

Not a new component. Not a new standard.
A decision asked for on purpose, made during execution, and made observable.

Putting it All Together: Key Points and Concluding Thoughts

Earlier, researchers had to adapt to systems. Now systems adapt to researchers.

1

Every OpenMP program already delegates its most important decisions

which **schedule**, **chunk**, task, *device*, **core**, how many teams — to layers that do not talk to one another.

2

At loop level we showed the decision can be adaptive

(**schedule**, **chunk**) per loop, per time step, from the same binary — and that nobody is told when it changes.

3

The same delegation happens at task, device, OS, and MPI

so **adaptivity** has to be **first-class**, **observable**, and **cross-layer**.

“One size does not fit all”

— and nothing tells you in advance which case you are in!



Performance, **portability**, **productivity** and **sustainability** are **not separate** problems but paid together, every time a piece of work is assigned to a resource by a decider nobody sees.

Who decides (from now on)? The OpenMP runtime, adaptively, in the open, together with its neighbors: asked to do so, reporting what it chose, in harmony with the OS and MPI.

The **scheduling decision** gets made either way. The question is whether **anyone can see and benefit it**.

In Memoriam: Johannes “Jojo” Doerfert, 1989–2026

LLVM since 2014 · OpenMP ARB since 2018 · code owner, LLVM OpenMP offloading · ANL → LLNL · IEEE-CS TCHPC Early Career Award 2023

“

Manual efforts to rewrite, tune, or adapt code are often signs for missing tools.

J. Doerfert's noble research goal: to help people exploit hardware to the fullest without having to become experts in the hardware or the software stack.



My personal gratitude: At the SPEC HPG meeting in 2019 in Chicago, he applauded our choice to build LB4OMP inside the LLVM OpenMP runtime, and offered to review our upstream patches. We were not fast enough...

Acknowledgments and Collaborators: Past, Present, and Future

HPC Group Basel · scheduling & OpenMP · MSc / BSc students · SPH-EXA & the science



UniBas current and former members and visitors



Florina Ciorba



Rubén Cabezon



Ali Mohammed



Ahmed Eleliemy



Danilo Guerrero



Aurelien Cavelan



Jonas H. Müller
Komdörfer



Michal Grabarczyk



Ayoub Neggaz



Thomas
Jakobsche



Osman Seckin
Simsek



Joseph Touzet



Helena Schmidt



Yiqing Zhu



Quentin Guilloteau



Reto
Krummenacher



José A. Escartín



Denisa
Constantinescu

CSCS



Sebastian Keller



Jean-Guillaume
Piccinali



Jean Favre



Jonathan Coles

UniZH



Lucio Mayer



Noah Kubli



Darren Reed

UP Catalunya



Axel Sanz

Scheduling & OpenMP, DMI-HPC in Basel

F. M. Ciorba · A. Mohammed · A. Eleliemy · J. H. Müller Komdörfer ·
O. S. Simsek · T. Jakobsche · Q. Guilloteau · R. Krummenacher · A. Cavelan ·
Y. Zhu · A. Boulmier

Scheduling & OpenMP collaborators

I. Banicescu · M. Klemm · V. Kale · C. Iwainsky · T. Ilsche · R. Schöne · M.
Schulz · S. J. Newcome (TUM) · A. Gentile · J. Brandt

MSc & BSc students whose work is in this talk

L. Kury · G.-A. Wetten · A. Birgovan · N. Shatri · R. Gondal · A. Jasari

SPH-EXA and the science

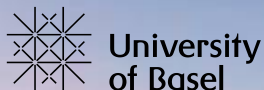
R. M. Cabezon · S. Keller · L. Mayer · D. Reed · N. Kubli · H. Schmidt · J.-G.
Piccinali · J. Favre · J. Coles · D. Garcia-Senz · A. Sanz · O. Avril · A. S. Lechuga
· R. Klessen, ...

Funding and allocations

SNSF · Innosuisse · SERI · PASC · SKACH · EU H2020 957407 (DAPHNE) ·
EuroHPC: LUMI-G Extreme Scale Access; Vega 2025D03-111 ·
CSCS: Alps / Daint, CHRONOS

Thank you for your interest in scheduling & resource management (decisions).

It takes a village. None of this was done by one person.



Thank you!
Questions and counter-
examples welcome 😊

Who Decides?

Beyond Runtime Defaults, Toward Adaptive Multi-Device OpenMP

Florina M. Ciorba

Joint Keynote II

IWOMP & EuroMPI 2026 · TU Wien, Vienna, Austria

October 8, 2026, 9:00–10:00

florina.ciorba@unibas.ch | [@cflorina.bsky.social](https://www.bsky.social/@cflorina) | www.linkedin.com/in/cflorina



OpenMP is really portable 😊