A Runtime Approach for Dynamic Load Balancing of OpenMP Parallel Loops in LLVM

University of Basel

Jonas H. Müller Korndörfer, Florina M. Ciorba, Akan Yilmaz
Department of Mathematics and Computer Science, University of Basel, Switzerland
jonas.korndorfer@unibas.ch, florina.ciorba@unibas.ch, akan.yilmaz@stud.unibas.ch

Christian Iwainsky
Technische Universität Darmstadt, Germany
christian.iwainsky@sc.tu-darmstadt.de

Johannes Doerfert, Hal Finkel
Argonne National Laboratory, Lemont, IL, USA
jdoerfert@anl.gov,hfinkel@anl.gov

Vivek Kale
Brookhaven National Laboratory, Upton, NY, USA
vkale@bnl.gov

Michael Klemm
Intel Deutschland GmbH, Feldkirchen, Germany
michael.klemm@intel.com

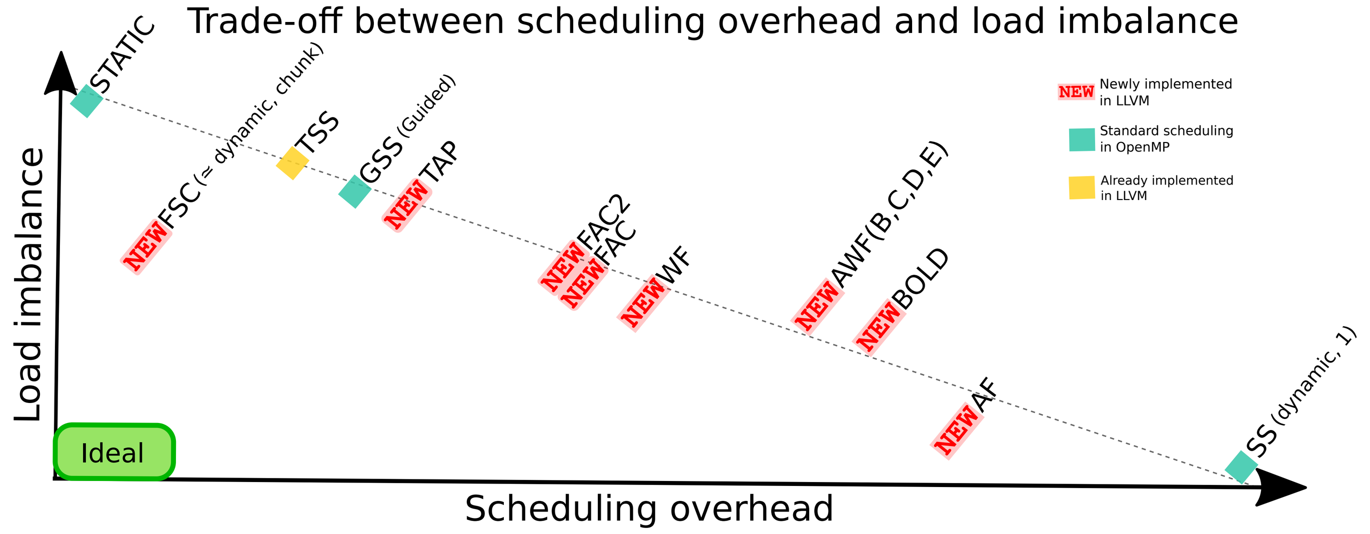
Swiss National Science Foundation

1. Research Problem

- **Load imbalance** is the major source of performance loss in computationally-intensive applications.
- **Efficient scheduling** of parallel loops can improve the performance of such programs.
- The **current OpenMP specification only provides three choices for loop scheduling** which do not support a wide research scenario with great potential for performance improvement.
- **This work augments the LLVM OpenMP runtime library** (RTL) with eleven state-of-the-art plus three improved and ready-to-use scheduling techniques.
- We provide a **comparison between the existing and the added loop scheduling techniques** on several applications from the NAS, SPEC OMP 2012, and CORAL2 benchmark suites.
- **Previous work has implemented a few non-adaptive dynamic scheduling techniques** [1, 2, 4]. However, no adaptive scheduling techniques were previously integrated into the OpenMP RTL.
- We verify the **overhead caused by the usage of the LLVM runtime** against the usage of the compiler for performing dynamic load balancing (DLB) of OpenMP parallel loops.

OpenMP 5.0

- Scheduling techniques available in the current OpenMP standard: `static`, `guided (GSS)`, and `dynamic (SS)`.
- Certain application-system scenarios require more sophisticated scheduling techniques yielding unexploited performance improvements.



Why LLVM?

- The LLVM OpenMP runtime (`llvm.org`) is open-source and widely used in many production and scientific codes.
- The LLVM OpenMP runtime library, **libomp**, is highly compatible with other OpenMP implementations, such as Intel and GCC.

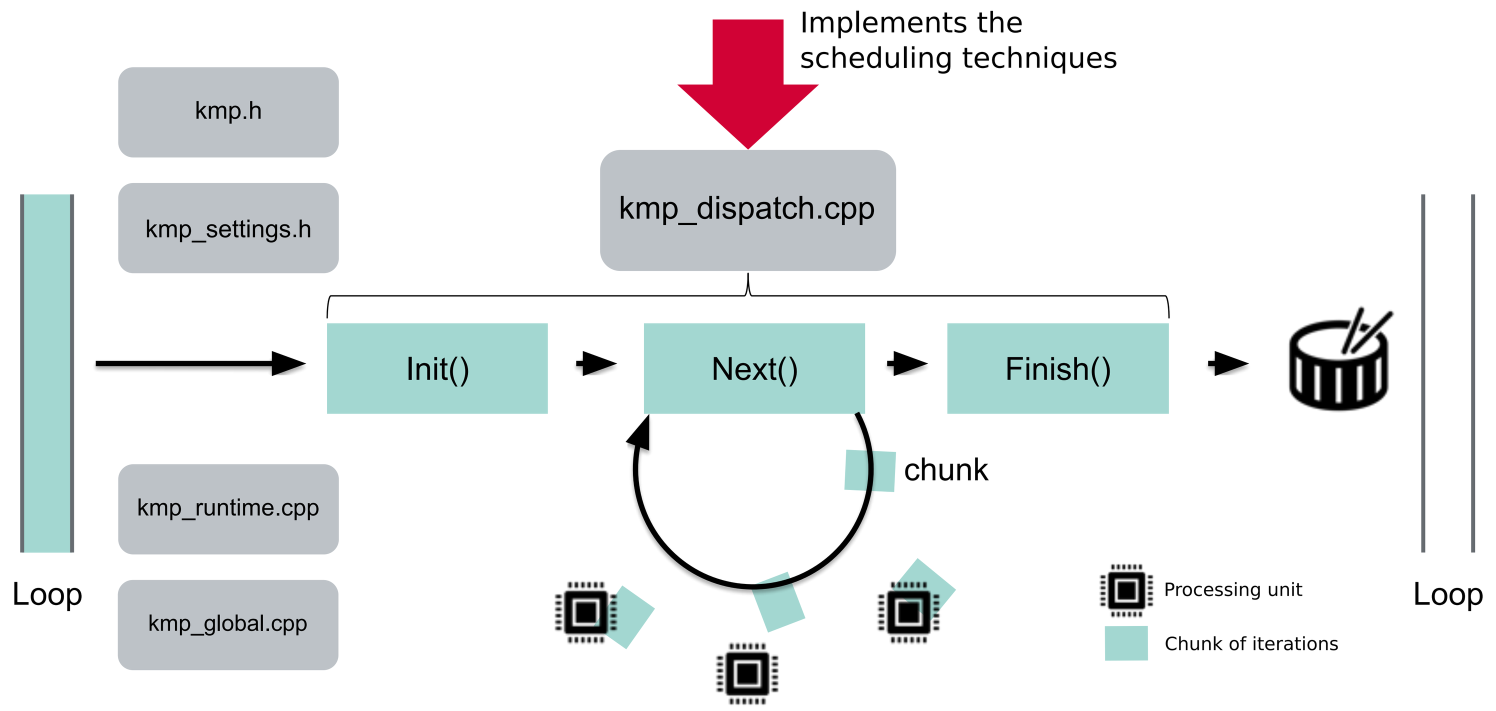
Scheduling Techniques Added to the LLVM OpenMP Runtime Library

Fixed Size Chunking (`fsc`), 1985
Factoring (`fac`), 1992
Factoring 2 (`fac2`), 1992
Taper (`tap`) - 1992
Weighted Factoring (`wf`), 1996
BOLD (`bold`), 1997
Adaptive Weighted Factoring and variants (`awf_b`, `awf_c`, `awf_d`, and `awf_e`), 1999-2002
Adaptive Factoring (`af`), 2000
Improvements to the `fac`, `fac2`, and `af` techniques are denoted by the suffix "`_a`", 2019

2. OpenMP Loop Scheduling in LLVM

Files Modified for the Implementation of Additional Scheduling Techniques

- **Libomp** uses three main functions to perform the scheduling of iterations from a loop onto threads: `init()`, `next()`, and `finish()`.
- The added loop scheduling techniques are located in the **kmp_dispatch.cpp** file.
- The remaining files initialize required environment variables and make the OpenMP runtime system aware of the eleven newly introduced techniques.



3. Design of Experiments

Details about the benchmarks and the modified loops.

Benchmark suite - App	Size	Modified loop file/line	Short Description
NAS 3.4 - MG	class C	mg.f/606	Approximate the solution to a three-dimensional discrete Poisson equation using the V-cycle multigrid method.
SPEC OMP 2012 - 350md	ref	int_ion_mix_direct.f/84/41/132, int_ion_pure_direct.f/85/42/142, int_nn_direct.f/117/55/186	Molecular dynamics
OmpSCR - md	16384 particles, 1000 steps	c_md.c/162	Molecular dynamics
CORAL2 - Quicksilver	10000000 particles	mc_omp_parallel_for_schedule_static.hh/4	Monte Carlo transport

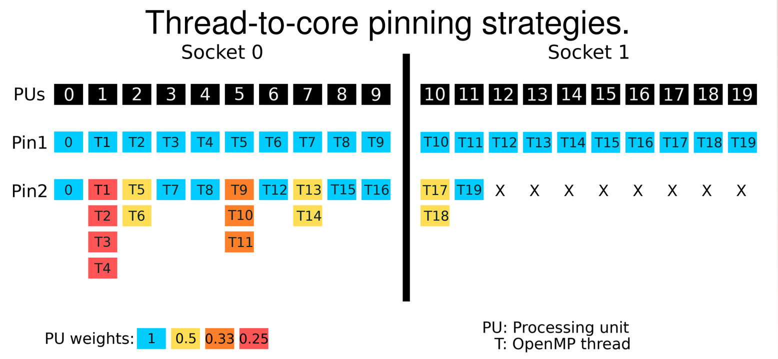
Overview of loop scheduling techniques.

Scheduling technique	Category			Chunk calculation		Chunk size		Use of batches	
		Dynamic							
	Static	Nonadaptive	Adaptive	Deterministic	Probabilistic	Fixed	Variable	Yes	No
Static block cyclic (STATIC)	✓			✓		✓			✓
Self-scheduling (SS - Dynamic)		✓		✓		✓			✓
Guided self-scheduling (GSS)		✓		✓			✓		✓
Trapezoid Self-Scheduling (TSS)		✓		✓			✓		✓
Fixed size chunking (FSC)		✓			✓	✓			✓
Factoring (FAC)		✓			✓		✓	✓	
Factoring 2 (FAC2)		✓			✓		✓	✓	
Taper (TAP)		✓		✓			✓		✓
Weighted factoring (WF)		✓			✓		✓	✓	
BOLD (BOLD)			✓		✓		✓	✓	
Adaptive weighted factoring (AWF-B)			✓		✓		✓	✓	
Adaptive weighted factoring (AWF-C)			✓		✓		✓		✓
Adaptive weighted factoring (AWF-D)			✓		✓		✓	✓	
Adaptive weighted factoring (AWF-E)			✓		✓		✓		✓
Adaptive factoring (AF)			✓		✓		✓		✓

System characteristics.

Node type	Intel Broadwell	4 nodes, 20 cores each
	Intel Xeon Phi KNL	1 node, 64 cores
Network	Topology	Two-level non-blocking fat-tree
	Fabric	Intel Omni-Path
	Bandwidth	100 Gbit/s
	Latency	100 ns

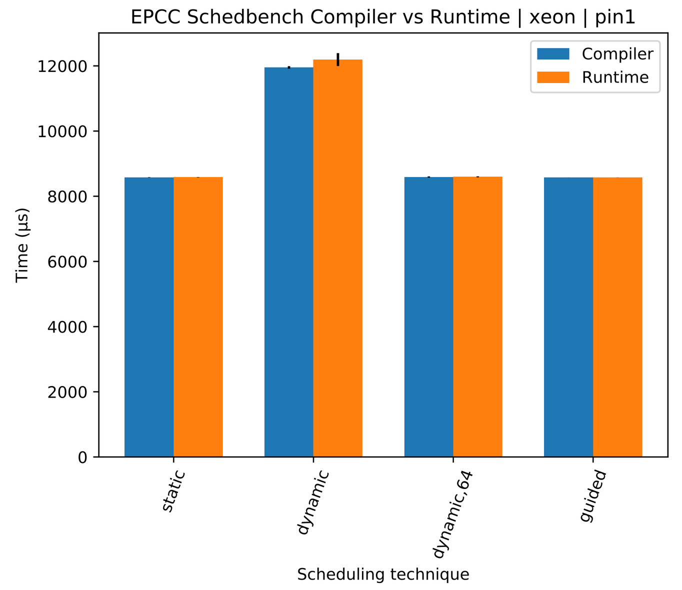
Thread-to-core pinning strategies.



4. Overhead of the Runtime DLB Approach

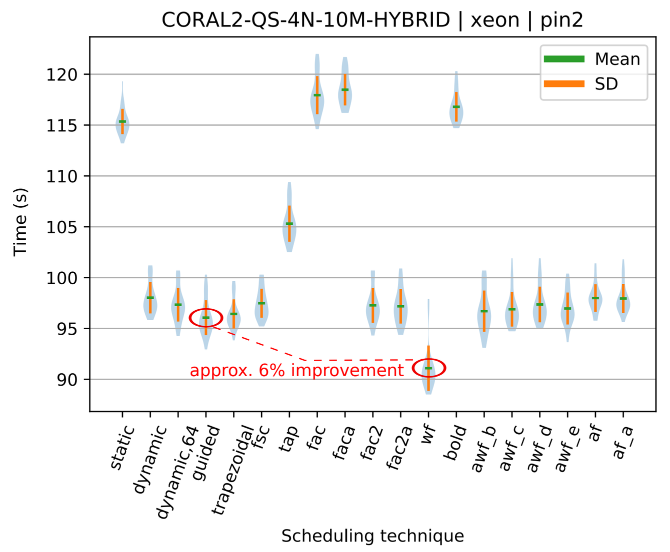
The runtime does not induce significant overhead!

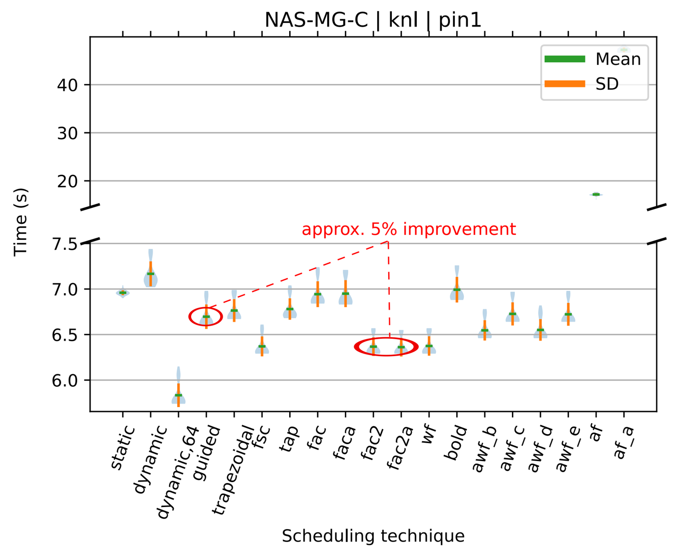
Comparison of the performance of the OpenMP standard techniques called by the runtime versus by the compiler. The EPCC benchmark was used in this experiment to measure the overhead induced by the runtime DLB approach, which shows to be insignificant.

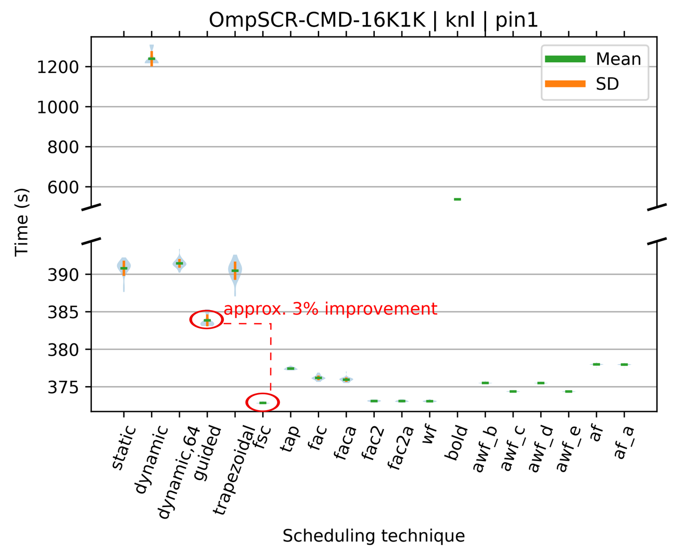


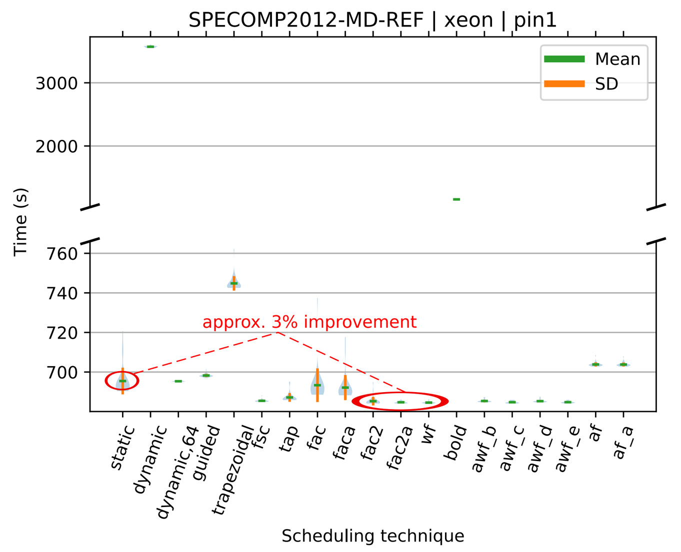
5. Performance Experiments

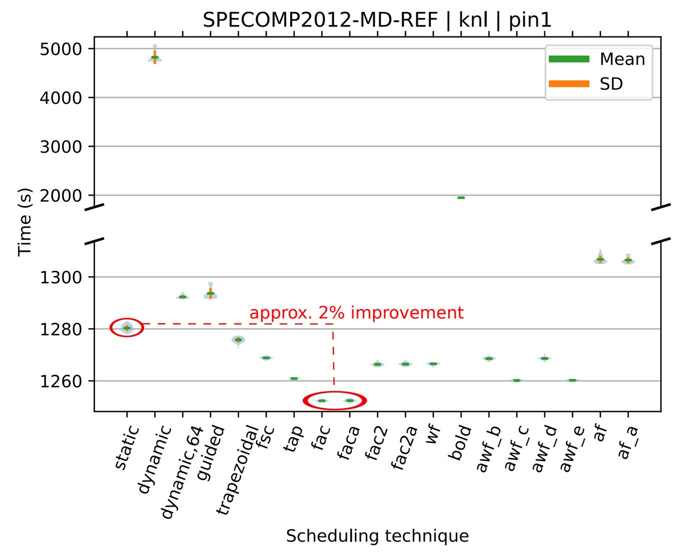
Parallel execution time of the 3 standard, 1 non-standard, and 11 (plus the 3 optimizations) added scheduling strategies on various benchmarks executed on two multicore systems. Each experiment was repeated 30 times. Green bars denote the mean, while orange bars denote the standard deviation of the parallel executions. Plot titles denote **benchmark suite - application - input size | system partition | pinning strategy**.

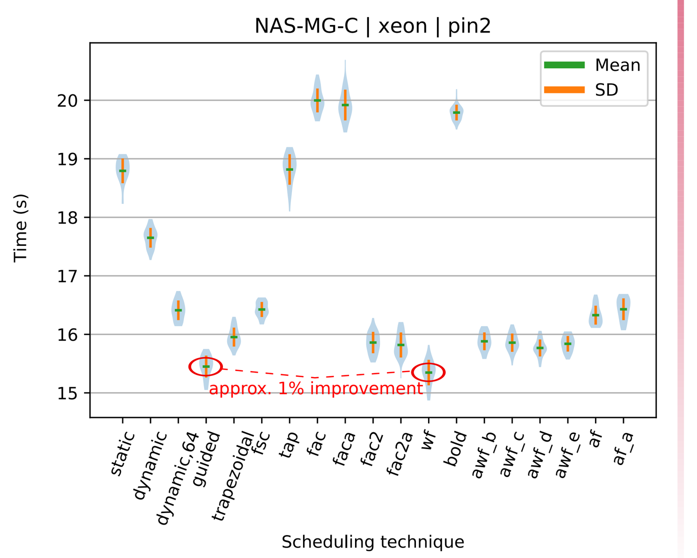












6. Conclusion

Performance gain beyond OpenMP standard

- OpenMP applications can improve beyond the performance delivered by the standard scheduling clauses.
- We provide a comparison between the standard scheduling techniques versus the eleven state-of-the-art plus three improved scheduling techniques.
- The performance gains depend on the degree of load imbalance during the application execution on a particular system.

Insignificant overhead

- Frequent calls to the OpenMP runtime library may cause overhead that can negate the well-known benefits of dynamic scheduling. The overhead measurements, however, show that such overhead is minimal for *static* and *guided* (GSS) while it is noticeable only for `dynamic` (SS).

Poorest performer

- The standard scheduling technique, `dynamic, 1` (or SS), rarely offers good performance. The performance degradation is due to the high overhead caused by frequent calls to the scheduling function and loss of data locality incurred by this technique.
- Further cache-aware experiments are needed for a precise chunk size setup for a particular application-system pair.

Consistent outperformer

- `fac2` is the most consistent and outperforming scheduling technique among the factoring methods.
  - `fac2` does not use mutual exclusion.
  - `fac2` does not require profiling information.

Information vs. performance trade-off

- Techniques such as `fsc`, `fac`, `tap`, and `bold` require precise application profiling information.
  - Performance degrades with `bold` when the profiling data is inaccurate.
  - `fsc` outperformed every technique with accurate input profiling data.
- The profiling process is time-consuming since it must be done prior to the application execution for a particular input on the particular system.

7. Future Work

User-Defined Scheduling Interface

- We will develop an interface that allows the user to define custom scheduling techniques that can be specialized to the application needs [3]. This will allow further performance improvements in many applications on a broad range of systems.

Upstream to LLVM

- We will upstream our implementation into the LLVM OpenMP runtime to make it available to the community. This will be done in individual patches, each with a single scheduling technique.

Acknowledgment

This work is in part supported by the Swiss National Science Foundation in the context of the "Multi-level Scheduling in Large Scale High Performance Computers" (MLS) grant, number 169123, and by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration.

References

[1] Bak, S., Guo, Y., Balaji, P. and Sarkar, V. Optimized Execution of Parallel Loops via User-Defined Scheduling Policies Proceedings of the 48th International Conference on Parallel Processing (ICPP 2019).
[2] Ciorba, F. M., Iwainsky, C. and Buder, P., 2018. OpenMP Loop Scheduling Revisited: Making a Case for More Schedules. Proceedings of the 2018 International Workshop on OpenMP (iWomp 2018).
[3] Kale, V. and Iwainsky, C. and Klemm, M. and Korndörfer Müller, J. H. and Ciorba, F., 2019. Towards A Standard Interface for User-Defined Scheduling in OpenMP Proceedings of the 2019 International Workshop on OpenMP (iWomp 2019).
[4] Kasielke, F., Tschüter, R., Velten, M., Ciorba, F. M., Iwainsky, C. and Banicescu, I. Exploring Loop Scheduling Enhancements in OpenMP: An LLVM Case Study. Proceedings of the 18th International Symposium on Parallel and Distributed Computing (ISPDC 2019).