Exploring Trade-offs of Compiler Optimizations to Enable Performance Portability for Multi-level Memory Hierarchies

Aleksei Levchenko

Peter the Great St.Petersburg Polytechnic University, Saint Petersburg, Russia
a@expx.org

Abstract. Performance portability problem is manifested for architectures with deep memory hierarchies, in particular, as a result of insufficient spatial locality support by compiler infrastructures. A polyhedral optimization approach can target spatial locality, but faces a number of challenges like an ambiguity in compatibility with other optimizations, a lack of polyhedral ready benchmarks and effects of non-uniformity of real world systems with a multi-level memory. Complementing the prior research of selecting optimizations, this paper focuses on experimental characterization of loop tiling and vectorization using the full proxy application. The presented approach makes the portability of performance provable for target architectures with deep memory hierarchies. To this end, large-scale ccNUMA macronodes are considered as experimental prototypes for hypothetical HPC designs of a capacity-bandwidth type, capable of imposing singular challenges for performance portability.

Keywords: Benchmarking, Locality, Loop tiling, Multi-level memory hierarchy, Performance portability, Polyhedral model, Program performance, ccNUMA

1 Introduction

In the run-up to the era of exascale computing, a forthcoming migration to advanced future systems involves a joint consideration of a notorious performance problem in connection with a performance portability issue. The performance portability challenge is manifested in an architecture-dependent performance degradation of a specialized version of code when compiling and running it across multi-core systems developed using new architectural principles that are different from the original ones. One of the predicted features of the next generation systems of capacity-bandwidth type is the existence of logically indivisible, globally addressable petabytes of RAM obtained by combining memories of a set of computing nodes [9]. The core of performance portability issue of the last class of systems remains the spatial locality that would almost likely be suboptimal for contemporary scientific applications. The polyhedral model is a promising and verifiable approach to achieve portability of performance over architectures with a hardly predictable spatial memory access behavior. An important piece in the puzzle is to evaluate the effect of a complex combination of polyhedral compilation techniques against the background of syntactic-level loop transformations already available [4,13,22]. Despite the growing number of papers on polyhedral frameworks development, there are, however, some issues associated with a lack of methods to achieve and estimate the performance portability, even with a well-known contemporary hardware. Accordingly, a number of steps might be proposed to clarify the methodology of performance improvement predictions based on compiler transformations.

Ad locum, this paper brings the following contributions. The presented approach allows to evaluate the performance portability of the code compiled using the relevant compilation algorithm for systems with different levels of a deep hierarchical memory. The first step is to define the set of target systems, i.e., ccNUMA macronodes. At the second stage ad hoc models should be formulated for the core parameters of performance portability, specifically spatial locality in the case at hand. The third stage involves the selection of the most promising compiler optimizations in conjunction with relevant HPC ready benchmarks. Further, the results of optimizations are compared with a reference model at the fourth experimental stage for architecturally affined systems with a deepening memory hierarchies. In the context of performance portability, a distinctive feature of proposed approach is a more reliable estimation of the trade-offs of compilation algorithms for current large-scale systems and a possibility of extrapolation of these results for hypothetical exascale designs.

The rest of this paper is organized as follows. Section 2 defines the set of target systems and dissects the existing background in achievement performance portability via compiler optimizations. Section 3 presents the core part of approach, which includes model considerations to evaluate the effect of optimizations. Section 4 is about selecting transformations for performance portability. Results of experimental evaluation and discussions are reported in Section 5. Further, Section 6 refers to the previous research, proving the interim findings of this paper and concerning the issues of locality transformations for performance portability towards multi-level memory and targeting spatial locality using polyhedral optimizations. Finally, Section 7 contains final remarks and discusses future work.

2 Background and Notation

At the first stage of the proposed approach, this section gives an idea about the targets, portability of performance over which is investigated. In the context of this work, a special subset of performance portability is implied when the performance of unoptimized code is ported to multi-level memory hierarchies of macronodes. Macronodes are shared memory multi-machine nodes with deep memory hierarchy based on Cache-Coherent Non-Uniform Memory Access (cc-NUMA) architecture. Table 3 enables to deduce an inference about the available macronodes configurations. Since the hypothetical systems of capacity-bandwidth type will have a depth complexity of globally addressable memory that is not comparable with a currently available one, the prototype of such system should have the approximate extreme characteristics. Being a shared memory clusters, current ccNUMAs are equipped with a larger amount of RAM and the on-line CPUs, than it is available on a typical general-purpose cluster node. The largest available multi-machine node, so-called *jumbonode* defined in paper [7], is capable of providing more than 12Tb of RAM at a time with 3K CPUs running a single operating system instance. At present, these features allow to extrapolate its performance for hypothetical characteristics of future machines [18]. Obviously, the main difference between the macronodes and the other get-at-able architectures is the possibility of a much larger indivisible amount of globally addressable memory controlled by one OS instance with a record number of on-line CPUs. It is this criterion has formed the basis for considering the ccNUMA architecture as an affordable prototype and a primary target since the lack of hypothetical machines that have not been developed yet. This viewpoint is grounded on a number of the previous works mentioned in the Section 6.

Deep memory hierarchy causes significant challenges for loop-intensive applications, well-functioning in a general-purpose HPC environment, but failing frequently in circumstances when spatial locality degrades [33]. In comparison with other software levels (e.g., performance portability libraries), the way of compiler analysis and optimizations is evidently more promising in terms of software/architecture codesign, due to better awareness of both the program and the underlying hardware. In particular, the polyhedral model, a mathematical framework to transform affine loop nests, is a promising way to achieve performance portability over targets with deep memory hierarchies via implementation of transformations as a single algebraic operation [2]. Compilers based on the polyhedral model provide alternative ways to use available resources of parallelism through analysis and transformations of the loop-based code. Properly implemented tiling, which reorganizes computation iteration space to improve cache reuse, can thus improve data locality and, as a consequence, the performance of iterative algorithms for a class of numeric programs. In the case of macronode, the tile size is critical, and it is prescribed by the cache/TLB/NUMA node size that requires a multi-aspect automatic determination of the optimal boundaries for the loop nest.

Therefore, it is argued that the deep memory hierarchy of macronodes specified above can demonstrate singular performance portability challenges that are difficult to meet in traditional computing clusters. While porting performance to macronodes, the estimation of improvements and drawdowns should be mapped to the prediction of performance models, pre-developed contextually.

3 Ad-Hoc Models

At the second stage, it is necessary to examine performance portability models for the considered targets (macronodes). Strict performance models can be efficient to characterize improvements provided by optimizers via several techniques. In this respect, the challenge looks as a number of limitations of the disparate models. A drawback here is that the cost models of compiler transformations often do not go beyond transformations in themselves, unlike the external memory-wide view of the entire memory subsystem. Meanwhile, it is possible to consider the main characteristics of the macronode that will affect the performance model of the proxy application. In this context, proxy application is a surrogate, representative scientific code for which there is a number of equivalent implementations using alternative parallel programming models and targeting multiple architectures. The aspects of using proxy applications implied in this paper are within the framework of the concept proposed in [14]. Eq. 1 associates computational proxy kernels, which the proxy application consists of, with a set of code improvements applied by the compiler simultaneously or in turn [27]. Let $Kernel_{i,opt,proxy}$ denotes the number of optimized proxy kernels, $T_{i,m\text{-}node,opt}$ is the execution time of proxy kernel i on macronode after applying every optimization *opt*. Under this assumption, the total execution time of proxy application is

$$T = \sum_{i=1}^{n} Kernel_{i,opt,proxy}\, T_{i,m\text{-}node,opt} \tag{1}$$

Next, performance portability PP of proxy kernel can be determined according to the formula 2, proposed by Pennycook et al. [25], and reinterpreted here for the case of macronodes:

$$PP_{m\text{-}node,opt,proxy} = \begin{cases} \dfrac{|M|}{\sum_{i \in M} \frac{1}{T(opt,proxy)}}, & \textit{if} \ \ \forall i \in M, \\ 0 & \textit{otherwise} \end{cases} \tag{2}$$

where $T(opt, proxy)$ denotes execution time of optimized proxy kernel for macronode i, $|M|$ — the set of macronodes. Here, spatial locality is considered as a key parameter of performance portability to overcome the notorious memory wall problem. Spatial locality reflects the tendencies in application behavior to access neighboring memory regions near regions that have been recently accessed. To consider the impact of compiler optimizations on spatial locality, it is necessary to analyze access to neighboring memory regions, which is the responsibility of the compiler. The results of proxy kernels can be compared using Eq. 3, that yields the locality measure previously defined by Dümmler et al. [8] using logarithmic geometric mean of access distances and reinterpreted here for macronodes as

$$L_{m-node,proxy} := \sum\nolimits_{\upsilon_s \in MV_s} \left(\sum_{i=1}^{l_{\upsilon_s}} \log_2 d_{s_i}(\upsilon_s) \right) \tag{3}$$

where $d_{s_i}(\upsilon_s)$ is spatial access distance of a variable $\upsilon_s \in MV_s$ (in the multiset of variables MV_s), l_{υ_s} is the total number of accesses to a variable $\upsilon_s \in MV_s$. The idea is to deduce the locality of computationally equivalent versions of the proxy application with different compiler optimizations for target macronodes.

As a result, the components of the performance portability equation, particularly the set of target systems and the main performance parameter (i.e., spatial locality), were identified. Additionally, the definition of the proxy kernel/program and the general formula of its execution time are given.

4 Selecting Transformations for Performance Portability

Even so, the next stage of the proposed approach involves the selection of the most promising transformations for the proxy application. Currently, the challenge is a lack of applications, that may be considered as polyhedral benchmarks as such. Polybench suite, for instance, is commonly used for this purpose [3, 5, 11]. However, systems with deep memory can achieve deceptively high levels of performance on small benchmarks, but lose performance in tasks of more realistic sizes. Therefore, of particular interest is a study of the polyhedral optimizations effects on behavior of a full-fledged scientific application in a real-world architecture, at least of a proxy application that can be divided into multiple proxy kernels. The experimental evaluation of optimized programs may be limited by the availability of a suitable benchmark code and the ability of a polyhedral optimizer to perform its transformations. As an example, the High Performance Conjugate Gradients (HPCG) Benchmark provides SpMV and symmetric Gauss-Seidel preconditioner with loop carried dependencies. HPCG can be considered as a proxy application, since it already has versions for different parallel programming models, namely MPI, OpenMP, SHMEM [1], etc. Although in this paper HPCG is still under consideration to be proxy application, run-time reordering transformations like sparse tiling that improve data locality in general have high potential for symmetric Gauss-Seidel kernel which dominates in the HPCG runtime.

Instead, a thoroughly studied proxy application, Livermore Unstructured Lagrange Explicit Shock Hydrodynamics (LULESH) has been used so far to evaluate the optimizations effects. LULESH solves one octant of the spherical Sedov blast problem using Lagrange hydrodynamics [24], which is representative of existing HPC codes and is able to demonstrate the complexity of poor spatial locality problem. This paper focuses on the traditional OpenMP implementation of LULESH as a consequence of the core architectural characteristics of the macronodes. In this respect, OpenMP programming model, the base version of LULESH alongside with serial and MPI code, is considered to be widely used mostly at the intra-node level. At the same time, it hypothetically can be used in hybrid MPI+OpenMP+X fashion which is considered as promising for exascale supercomputer designs. The presence of more than 1K of threads living within the terabytes of shared memory is a great stress-test leastwise against the background of known benchmarking efforts.[†] The performance of subsequent LULESH implementations for emerging programming models is often compared by researchers with the characteristics of OpenMP code. Currently, LULESH results are known for target architectures like BG/Q [19], Cray XE6 [23], Power 7, AMC [12], etc. Another advantage of focusing on the traditional OpenMP programming model is the support by number of polyhedral infrastructures and corresponding libraries.

As mentioned above, while Polybench suite is specially designed to contain predefined static control parts (SCoPs), LULESH is a more realistic, full proxy ap-

[†] Up to 1024 threads supported using customized OpenMP implementation.

plication, but it is not polyhedral benchmark from the cradle. It provides relatively more complex SCoPs, accordingly, it has to be prepared to become polyhedral optimizable. The range expansion of traditional polyhedral benchmarks here is a consequence of the search for applications like HPCG or LULESH containing important computational proxy kernels, which (1) would be representative of wide range of important scientific applications, and (2) would allow to select simplified tasks from the full proxy application, enabling the semi-automatic iterative selection of compiler optimizations. The most significant modifications of OpenMP implementation of LULESH include resolving indirect array accesses. Potential SCoPs can be formed by converting from the most time-consuming large parallel OpenMP regions. The limitation here is that SCoPs contain multiple redundant dependencies between various statements, which must be eliminated. This approach was proposed by Wang et al. [32], where the variants of LULESH code were generated using PoCC (the Polyhedral Compiler Collection) [29]. The list of optimizations applied in this paper or considered for LULESH in the well-known studies beyond the scope of this work is shown in Table 1.

Table 1. List of optimizations that have been applied (+) to LULESH in this paper in the context of prior work considerations

Possible optimizations	CalcKinematicsForElems	Hexahedron volume calculation
Array contraction [15]	Considered, applicable	Considered, applicable
Global allocation [19]	Considered, applicable	Considered, applicable
Loop fusion [15]	Considered, applicable	Considered, applicable
Loop distribution [12]	Considered, applicable	Considered, not applicable
(+) Tiling	Applied	Applied
(+) Vectorization	Applied	Applied

Loop tiling for data locality is an important addition in the context of the transformations already reviewed, because the right tile size/shape can take into account the sophisticated characteristics of a non-uniform deep memory. For example, when optimizing performance of a local memory, kernel loop can be produced exclusively for the local memory access and bounding loop can be produced to transfer the DMA operations outside the kernel loop in conditions of insufficient hardware coherence support for a local and global memory. Loop tiling requires development of the models in compatibility with other loop transformations.

5 Experimental Evaluation and Discussions

The fourth stage includes optimizations of the proxy program and the measurement of a number of metrics, the most important of which is the spatial locality. To this end, this paper uses Polly, a tool for the polyhedral optimization of the LLVM-IR for a data locality and parallelism [11]. Polly was used to detect SCoPs in canonicalized code in the front end that can be translated to a polyhedral representation and subjected to optimization. Optimizations, namely loop tiling and vectorization, were described manually in JSCoP format, which is specific to Polly, and applied through import/reimport mechanism of the polyhedral representation with modified schedules of the statements (Figure 1). Finally, the transformed polyhedral representation is used for the OpenMP code generation.

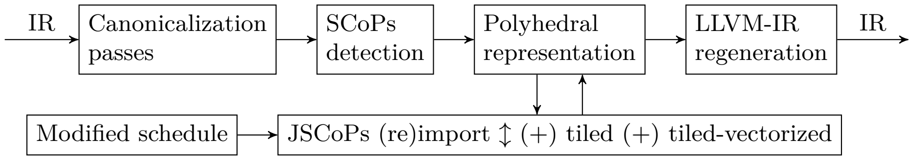


Fig. 1. Performing of manual optimizations on the polyhedral representation (JSCoPs with modified schedules of statements) using LLVM/Polly (re)import mechanism [11]

Experimental runs were carried out using available ccNUMA macronodes configurations (Table 3), and the average results are reported. The *Grind Time* metric, a measurement of the per-element compute time reported by LULESH, was used to compare early-stage results with reference baseline OpenMP code, where no SCoPs detection and code generation were used. Lower values of a Grind Time metric indicate an increase in performance. Table 2 compares the preliminary results of the optimized code with unoptimized version (NoOpt) and demonstrates that transformed LULESH exhibits superior *Grind Time* to a reference code. As shown in Table 3, in the case of 3TB macronode, the stated optimizations reduced LLC and TLB misses, and the percentage of vectorized floating point operations has been increased.

Figure 2 shows the results for the optimized and reference implementation of LULESH for a 90^3 problem. When applying *tiling+vectorization*, there is an improvement in performance due to NUMA aware allocation over the intra-macronode level when switching *Minimal* (752Gb) to *Medium* (3Tb). Although no special compiler optimizations have been applied to improve locality for HPCG, the results of HPCG obtained in paper [7] with some HPCG optimizations first described in [1] along with the models that take into account the characteristics

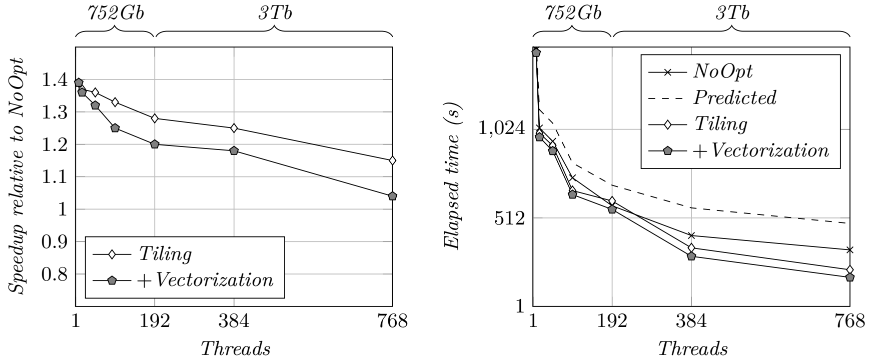
Table 2. Grind Time for considered versions of LULESH on target ccNUMA system (lower is better)

Applied optimizations	Grind time (μs/z/c)
NoOpt (baseline OpenMP code)	3.65
(+) Tiled	3.28
(+) Vectorized	3.03

Table 3. The effect of optimizations on vectorization and LLC/TLB misses across 752Gb and 3Tb macronodes

Target system	Single OS instance macronodes		
characteristics	Minimal	Medium	Jumbonode
Architecture details			
RAM	752Gb	3Tb	12Tb
NUMA node(s)	24	96	384
Board/Socket/Core(s)	4/12/192	16/48/768	64/192/3072
Description of improvements made			
FLOP Vectorization	+20%	+20%	Future work
LLC cache misses	-25%	-18%	Future work
TLB misses	-18%	-8%	Future work

of the macronode were used additionally to predict the approximate scaling of LULESH during aggregation of macronode memory to 3Tb of RAM.



(a) Scaling (relative to NoOpt)

(b) Timing results (predicted vs. NoOpt vs. optimized)

Fig. 2. Scaling/performance gain for LULESH with varying threads for a 90^3 sized mesh on the macronodes with 752Gb and 3Tb of RAM

Figure 3 illustrates the results of spatial locality measurement for multi-threaded LULESH for $30^3 \ldots 90^3$ sized mesh on the macronodes with 752Gb and 3Tb of globally addressable memory. Using the previously presented model, tiled version shows the expected better spatial locality compared to NoOpt version, which approximately matches the reported values of *Grind Time*, as well as the *Elapsed time* results. The main interest is the measurement of locality for the 3TB macronode, which is the maximum value in these experiments. For all problems, the spatial locality is better (lower) for the optimized code. The surface bundle is approximately the same for the minimum and maximum problem size.

Hence, in accordance with the previously considered definitions, the performance portability was achieved for macronodes under consideration.[††]

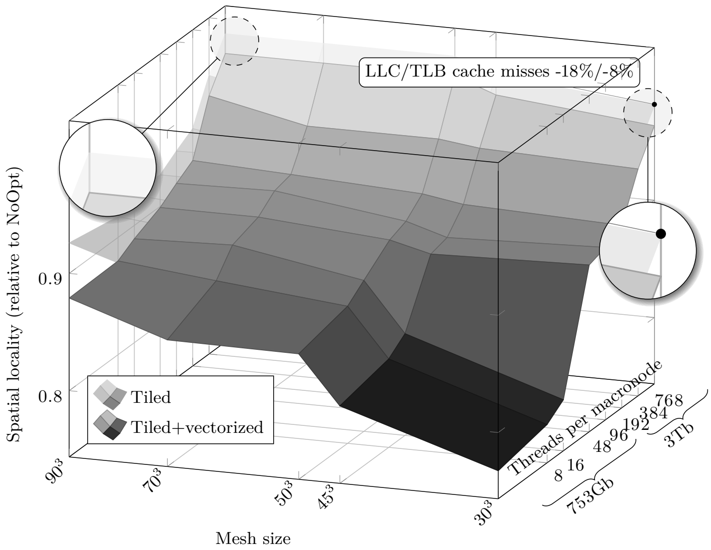


Fig. 3. Spatial locality (relative to NoOpt) for multithreaded LULESH with $30^3 \ldots\ 90^3$ sized mesh on the macronodes with 752Gb and 3Tb of shared memory (lower is better)

Regarding the trade-offs between the optimizations under consideration in terms of spatial locality and parallelism, loop fusion is also widely considered to improve locality due to a data motion reduction. Fusion reduces 45 loops to 12 loops, and number of OpenMP parallel regions is reduced from 30 to 12 [16]. One aspect of this is that as parallelism being increased, the redundant loop fusion may not properly use a hardware prefetching, and spatial locality will degrade. Agglomeration of the loop fusion tends to increase a number of dependencies that must be satisfied, as it appears in the case of fused LULESH. At the same time, the loop fusion can prevent proper parallelization of the OpenMP code, because the number of dependencies that need to be satisfied in the fused loops will grow. This factor should be taken into account when a fused-tiled implementation is used.

6 Related Work

The fundamental concepts of performance portability implied in this paper are closely adjacent to the terms used in dissertation [30] where the special

[††] Studies for hybrid OpenMP/MPI version of LULESH on the 12Tb Jumbonode now left for near future work.

optimization issues are considered in the context of massively threaded systems. At the same time, work [20] proposed profitable compiler optimizations for performance portability of CFD applications on multiple HPC systems. The subsequent work [28] clearly concludes that solution of the problem of analytical determination of tile size limits for loop tiling will help improve performance for a wide range of systems. The most detailed analysis of the concept of performance portability as such was proposed in work [25] and confirmed by the results of the study [17]. Papers [2] and [33] demonstrate the potential of the polyhedral model to achieve the portability of performance, in particular, due to the architectual modeling of spatial effects in the latter research report.

The conventional consideration of NUMA equipped deep memory systems as a prototype during the adaptation to exascale supercomputer designs is an inherent continuation of several recent works [10,18,26,35,36] and particularly the paper [6] that directly uses proxy applications for idem. In supercomputing circles, LULESH as a proxy application has been used in the research [12] on the codesign of compiler and Active Memory Cube (AMC), recently developed Processing in Memory (PIM) architecture for exascale computing, and in a study on the compiler optimizations selection, particularly for BG/Q by León et al. [19]. As for the currently known studies on polyhedral-related optimizations, LULESH was used in [21] for the study of tiled Concurrent Collections (CnC) implementation, superior to performance of traditional OpenMP implementation. Wang et al. [32] used LULESH to evaluate various optimizations including loop fusion and auto-parallelization of OpenMP baseline implementation, and the result demonstrates the possibility of using LULESH to characterize the performance improvement provided by the newly-developed polyhedral techniques. Last but not least, Verdoolaege et al. [31] provides an insight into targeting spatial locality via polyhedral scheduling using Pluto, and Zinenko et al. [34] proposes an algorithmic template capable of modeling the temporal/spatial locality of multiprocessors.

7 Conclusions and Future Work

The proposed approach allows to achieve the performance portability over a set of affined targets, that differ significantly in the number of levels of hierarchical memory. At the same time, this paper does not address the issue of performance portability from traditionally used massively parallel architectures to ccNUMA macronodes, which almost certainly will be suboptimal for a class of developed numerical software based on traditional concepts of a spatial locality. The above-mentioned fundamental works [20, 28, 30] give a sense of conceivable solution complexity. One aspect of this is that the issue of a backward performance portability from ccNUMA architecture to a traditional symmetric multiprocessor and massively parallel architectures will not be as acute as in the case of direct migration of performance from standard cluster to architectures with a large number of levels of memory hierarchy (i.e., to ccNUMA macronode).

Future work will investigate the opportunities for development of loop tiling performance models to improve fast algorithms to predict performance and automatic tile size selection. The computational kernels of particular significance include important stationary iterative methods such as Jacobi, Gauss-Seidel, SOR-like methods that are used as subroutines by other algorithms, e.g., in symmetric multigrid. Another idea being explored is to model fusion+tiling effects for this complex algorithms when porting performance to largest macronodes.

Acknowledgments. This work was financially supported by the Ministry of Education and Science of the Russian Federation in the framework of the state assignment No. 2.9517.2017/8.9 (the project theme "Methods and technologies for verification and development of software for modeling and calculations using HPC platform with extramassive parallelism").

References

1. Agarkov, A., Semenov, A., Simonov, A.: Optimized implementation of HPCG benchmark on supercomputer with "Angara" interconnect. In: Voevodin, V., Sobolev, S. (eds.) Proceedings of the 1st Russian Conference on Supercomputing — Supercomputing Days 2015. CEUR Workshop Proceedings, vol. Vol-1482, pp. 294–302. Research Computing Center, Moscow State University, Moscow, Russia (Sep 2015)
2. Benabderrahmane, M.W., Pouchet, L.N., Cohen, A., Bastoul, C.: The polyhedral model is more widely applicable than you think. In: Proceedings of the 19th Joint European Conference on Theory and Practice of Software, International Conference on Compiler Construction. pp. 283–303. CC'10/ETAPS'10, Paphos, Cyprus (Mar 2010)
3. Bielecki, W., Palkowski, M., Skotnicki, P.: Generation of parallel synchronization-free tiled code. Computing 100(3), 277–302 (Mar 2018)
4. Chatarasi, P., Sarkar, V.: Extending polyhedral model for analysis and transformation of OpenMP programs. In: 2015 International Conference on Parallel Architecture and Compilation (PACT). pp. 490–491 (Oct 2015)
5. Chatarasi, P., Shirako, J., Kong, M., Sarkar, V.: An extended polyhedral model for SPMD programs and its use in static data race detection. In: Ding, C., Criswell, J., Wu, P. (eds.) Languages and Compilers for Parallel Computing. pp. 106–120. Springer International Publishing, Cham (2017)
6. Cicotti, P., Mniszewski, S.M., Carrington, L.: An evaluation of threaded models for a classical MD proxy application. In: 2014 Hardware-Software Co-Design for High Performance Computing. pp. 41–48 (Nov 2014)
7. Drobintsev, P., Kotlyarov, V., Levchenko, A., Petukhov, E.: Experimental considerations towards effective memory bandwidth evaluation on large-scale ccNUMA systems. In: Sozykin, A., Ustalov, D. (eds.) Proceedings of the 3rd Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists. CEUR Workshop Proceedings, vol. Vol-1990, pp. 40–49. Krasovskii Institute of Mathematics and Mechanics, Russia, Yekaterinburg, Russia (Oct 2017)
8. Dümmler, J., Rauber, T., Rünger, G.: Combining measures for temporal and spatial locality. In: Min, G., Di Martino, B., Yang, L.T., Guo, M., Rünger, G. (eds.) Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops. pp. 697–706. Springer Berlin Heidelberg, Berlin, Heidelberg (Dec 2006)

9. Eisymont, L.: Hybrid strategy development of the supercomputer components. Open Systems. DBMS 25(2), 8–11 (Jun 2017)
10. Goglin, B.: Memory footprint of locality information on many-core platforms. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 1283–1292 (May 2018)
11. Grosser, T., Groesslinger, A., Lengauer, C.: Polly — performing polyhedral optimizations on a low-level intermediate representation. Parallel Processing Letters 22(04) (2012)
12. Jacob, A., Nair, R., Chen, T., Sura, Z., Kim, C., Bertolli, C., Antao, S., O'Brien, K.: Progressive codesign of an architecture and compiler using a proxy application. In: 2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). pp. 57–64 (Oct 2015)
13. Jing, M., Kong, F., Jin, X., Zeng, X.: An improved automatic parallelizing algorithm based on polyhedral model. In: 2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT). pp. 235–237 (Oct 2016)
14. Karlin, I., Bhatele, A., Keasler, J., Chamberlain, B.L., Cohen, J., Devito, Z., Haque, R., Laney, D., Luke, E., Wang, F., Richards, D., Schulz, M., Still, C.H.: Exploring traditional and emerging parallel programming models using a proxy application. In: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing. pp. 919–932 (May 2013)
15. Karlin, I., McGraw, J., Gallardo, E., Keasler, J., León, E.A., Still, B.: Memory and parallelism tuning exploration using the LULESH proxy application. In: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis. pp. 1429–1429 (Nov 2012)
16. Karlin, I., et al.: LULESH programming model and performance ports overview. Tech. Rep. LLNL-TR-608824, Lawrence Livermore National Laboratory (Dec 2012)
17. Kirk, R.O., Mudalige, G.R., Reguly, I.Z., Wright, S.A., Martineau, M.J., Jarvis, S.A.: Achieving performance portability for a heat conduction solver mini-application on modern multi-core systems. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). pp. 834–841 (Sep 2017)
18. Kotlyarov, V., Drobintsev, P., Levchenko, A., Voinov, N.: Adapting software applications to hybrid supercomputer. In: Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia. pp. 6:1–6:5. CEE-SECR '17, St. Petersburg, Russia (Oct 2017)
19. León, E.A., Karlin, I., Grant, R.E.: Optimizing explicit hydrodynamics for power, energy, and performance. In: 2015 IEEE International Conference on Cluster Computing. pp. 11–21 (Sep 2015)
20. Lin, P.H.: Performance portability strategies for computational fluid dynamics (CFD) applications on HPC systems. Ph.D. thesis, University of Minnesota (Jun 2013)
21. Liu, C., Kulkarni, M.: Evaluating performance of task and data coarsening in concurrent collections. In: Ding, C., Criswell, J., Wu, P. (eds.) Languages and Compilers for Parallel Computing. pp. 331–345. Springer International Publishing, Cham (2017)
22. Meeus, W., Stroobandt, D.: Data reuse buffer synthesis using the polyhedral model. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 26(7), 1340–1353 (Jul 2018)
23. Milthorpe, J., Grove, D., Herta, B., Tardieu, O.: Exploring the APGAS programming model using the LULESH proxy application. Tech. Rep. IBM Research Report. RC25555 (WAT1509-050), IBM Research Division, Thomas J. Watson Research Center (Sep 2015)

24. Padoin, E.L., Pilla, L.L., Castro, M., Navaux, P.O.A., Méhaut, J.F.: Exploration of load balancing thresholds to save energy on iterative applications. In: Barrios Hernández, C.J., Gitler, I., Klapp, J. (eds.) High Performance Computing. pp. 76–88. Springer International Publishing, Cham (2017)
25. Pennycook, S., Sewall, J., Lee, V.: Implications of a metric for performance portability. Future Generation Computer Systems (2017), https://doi.org/10.1016/j.future.2017.08.007
26. Perarnau, S., Zounmevo, J.A., Dreher, M., Essen, B.C.V., Gioiosa, R., Iskra, K., Gokhale, M.B., Yoshii, K., Beckman, P.: Argo NodeOS: Toward unified resource management for exascale. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 153–162 (May 2017)
27. Saavedra, R.H., Smith, A.J.: Performance characterization of optimizing compilers. IEEE Transactions on Software Engineering 21(7), 615–628 (Jul 1995)
28. Sharma, K.: Locality transformations of computation and data for portable performance. Ph.D. thesis, Rice University (Aug 2014)
29. Shirako, J., Pouchet, L.N., Sarkar, V.: Oil and water can mix: An integration of polyhedral and AST-based transformations. In: SC14: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 287–298 (Nov 2014)
30. Stratton, J.: Performance portability of parallel kernels on shared-memory systems. Ph.D. thesis, University of Illinois at Urbana-Champaign (2013)
31. Verdoolaege, S., Isoard, A.: Extending Pluto-style polyhedral scheduling with consecutivity. In: Proceedings of the 8th International Workshop on Polyhedral Compilation Techniques, IMPACT 2018. Manchester, United Kingdom (Jan 2018)
32. Wang, W., Cavazos, J., Porterfield, A.: Energy auto-tuning using the polyhedral approach. In: Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques, IMPACT 2014. Vienna, Austria (Jan 2014)
33. Zinenko, O., Verdoolaege, S., Reddy, C., Shirako, J., Grosser, T., Sarkar, V., Cohen, A.: Unified polyhedral modeling of temporal and spatial locality. Research Report RR-9110, Inria Paris (Nov 2017), https://hal.inria.fr/hal-01628798
34. Zinenko, O., Verdoolaege, S., Reddy, C., Shirako, J., Grosser, T., Sarkar, V., Cohen, A.: Modeling the conflicting demands of parallelism and temporal/spatial locality in affine scheduling. In: Proceedings of the 27th International Conference on Compiler Construction. pp. 3–13. CC 2018, Vienna, Austria (Feb 2018)
35. Zou, P., Allen, T., Claude H. Davis IV, Feng, X., Ge, R.: CLIP: Cluster-level intelligent power coordination for power-bounded systems. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). pp. 541–551 (Sep 2017)
36. Zounmevo, J.A., Perarnau, S., Iskra, K., Yoshii, K., Gioiosa, R., Van Essen, B.C., Gokhale, M.B., Leon, E.A.: A container-based approach to OS specialization for exascale computing. In: 2015 IEEE International Conference on Cloud Engineering. pp. 359–364 (Mar 2015)