Performance Improvement of Stencil Computations for Multi-core Architectures based on Machine Learning

Víctor Martínez, Fabrice Dupros, Márcio Castro, Philippe O.A. Navaux

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 305–314

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Performance Improvement of Stencil Computations for Multi-core Architectures based on Machine Learning

Víctor Martínez[1], Fabrice Dupros[2], Márcio Castro[3], and Philippe Navaux[1]

[1] Informatics Institute (INF), Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
{victor.martinez, navaux}@inf.ufrgs.br
[2] BRGM, Orléans, France
f.dupros@brgm.fr
[3] Department of Informatics and Statistics (INE), Federal University of Santa Catarina (UFSC), Florianópolis, Brazil
marcio.castro@ufsc.br

Abstract

Stencil computations are the basis to solve many problems related to Partial Differential Equations (PDEs). Obtaining the best performance with such numerical kernels is a major issue as many critical parameters (architectural features, compiler flags, memory policies, multithreading strategies) must be finely tuned. In this context, auto-tuning methods have been extensively used to improve the overall performance. However, the complexity of current architectures and the large number of optimizations to consider reduce the efficiency of this approach. This paper focuses on the use of Machine Learning to predict the performance of stencil kernels on multi-core architectures. Low-level hardware counters (e.g. cache-misses and TLB misses) on a limited number of executions are used to build our predictive model. We have considered two different kernels (7-point Jacobi and seismic wave modelling) to demonstrate the effectiveness of our approach. Our results show that performance can be predicted and that the best input configuration for stencil problems can be obtained by simulations of hardware counters and performance measurements.

Peer-review under responsibility of the scientific committee of the International Conference on Computational Science

Keywords: machine learning, stencil computation, multi-core, performance model

1 Introduction

Stencil computations lie at the heart of many problems in areas as diverse as electromagnetics, fluid dynamics or geophysics. The trend for High Performance Computing (HPC) applications is to pay a higher cost in order to optimize the overall performance. This comes from the complexity of many interdependent factors (non-uniform memory access, vectorization, compiler optimizations, memory policies) at an architectural level that may severely influence the application's behavior. This is particularly true for stencil numerical kernels that are usually

1877-0509
Peer-review under responsibility of the scientific committee of the International Conference on Computational Science
10.1016/j.procs.2017.05.164

memory-bound. Although a large body of literature on the optimization of this class of applications is available, predicting the performance on current architectures remains a challenge.

On the one hand, application tuning represents the classical methodology to squeeze the performance on multi-core architectures. Unfortunately, this approach leads to the exploration of a huge set of parameters, thus limiting its interest on complex platforms. In this context, several heuristics or frameworks have been proposed to speed up the process of finding the best configuration for stencil applications [3, 13, 15].

On the other hand, Machine Learning (ML) is a comprehensive methodology for optimization that could be applied to find patterns on a large set of input parameters. Recently, ML algorithms have been used on HPC systems under different situations. In [2] the authors proposed an ML-based approach to automatically infer a suitable thread mapping strategy for a given application. In [17] the authors used ML algorithms to select the best job scheduling algorithm on heterogeneous platforms whereas in [1] the authors proposed an ML-based scheme to select the best I/O scheduling algorithm for different applications and input parameters.

In this paper, we describe the procedure to build a suitable ML-based performance model for two classical numerical kernels: 7-point Jacobi and seismic wave modelling. This model allows us to simulate the performance behavior of stencil computations on multi-core architectures. The proposed model can be integrated in auto-tuning frameworks to find the best configuration for a given stencil application.

The paper is organized as follows. Section 2 provides the fundamentals of stencils under study. Section 3 describes the methodology of our ML-based approach. Section 4 presents experiment configuration, simulation performance, and model accuracy. Finally, Section 5 describes related works, and Section 6 concludes this paper.

2 Stencil models

From the numerical analysis point of view, stencil-based computations often arise when discretizing Partial Differential Equations (PDEs). For instance, the Finite-Difference Methods (FDMs) computational procedure consists in using the neighboring points in the north-south, east-west and forward-backward directions to evaluate the current grid point in the case of a three-dimensional Cartesian grid. The algorithm then moves to the next point applying the same stencil computation until the entire spatial grid has been traversed. The number of points used in each direction depends on the order of the approximation. In this context, a standard metric available to characterize a stencil kernel is the Arithmetic Intensity (AI), which is a measure of floating-point operations (FLOPS) performed by a given code section relative to the amount of memory accesses (bytes) that are required to support those operations. In this work, we study two well-known stencil kernels:

1. **7-point Jacobi**: This numerical kernel is known to be severely memory-bound as we need seven reads to compute and write the current grid point. The lower-bound of AI of this kernel is 0.18.

2. **Seismic Wave**: This numerical kernel corresponds to the discretization of the elastodynamics equation and is of great importance both for seismic hazard assessment as well as for the oil and gas industry. In our case, we consider a standard fourth order in space and second order in time approximation. This algorithm corresponds to the evaluation of six stress components (three in the diagonal direction and three off-diagonal) and 3 velocity components. Algorithm 1 provides a synthetic view of the computation of one of

Algorithm 1: Pseudo-code of the stress component (σ_{xx}) in the Seismic Wave kernel.

for $i = 1$ to N_x **do**
  for $j = 1$ to N_y **do**
    for $k = 1$ to N_z **do**
      $\sigma^{n+1}_{xx}(i,j,k) = \sigma^{n}_{xx}(i,j,k)$
      $+A_1[a_1(V^n_x(i+\frac{1}{2},j,k) - V^n_x(i-\frac{1}{2},j,k)) + a_2(V^n_y(i,j+\frac{1}{2},k) - V^n_y(i,j-\frac{1}{2},k)) +$
      $a_3(V^n_z(i,j,k+\frac{1}{2}) - V^n_z(i,j,k-\frac{1}{2}))]$
      $+B_1[a_1(V^n_x(i+\frac{3}{2},j,k) - V^n_x(i-\frac{3}{2},j,k)) + a_2(V^n_y(i,j+\frac{3}{2},k) - V^n_y(i,j-\frac{3}{2},k)) +$
      $a_3(V^n_z(i,j,k+\frac{3}{2}) - V^n_z(i,j,k-\frac{3}{2}))]$
    end for
  end for
end for

the diagonal components, where i, j, k represent a tensor field component in Cartesian coordinates (x, y, z), and V and σ represent the velocity and stress fields, respectively. The overall procedure is composed of two consecutive triple nested loops (stress and velocity) with interdependence between the computation of the components. The arithmetic intensity is close to 1.30 in this case. A detailed description of the numerical modeling of seismic waves on multi-core platforms is presented in [8, 12].

2.1 Implementation

We evaluate two distinct implementations of the previously described kernels:

1. **Naive:** This implementation corresponds to the standard exploitation of the triple nested loops coming from the three spatial dimensions. This allows for a straightforward usage of OpenMP directives to parallelize the computation.

2. **Blocking:** This implementation exploits space tiling techniques. The main idea is to exploit the inherent data reuse available in the triple nested loops of the kernel by ensuring that data remains in cache across multiple uses.

The naive version allows to validate our ML-based methodology in a straightforward way as the set of parameters to be tuned is reduced. The blocking version is much more challenging because finding the best shape or size for the tiles remains an open research topic [11]. We reuse insights and optimizations described in [8] in order to build robust implementations.

3 Machine Learning Methodology

In this section we describe our ML methodology which relies on support vector machines (SVM). First, we present the feature vectors considered in our study and we discuss how they are supported by SVMs. Finally, we describe our ML model.

3.1 Feature vectors

We considered three sets of vectors, which are described below:

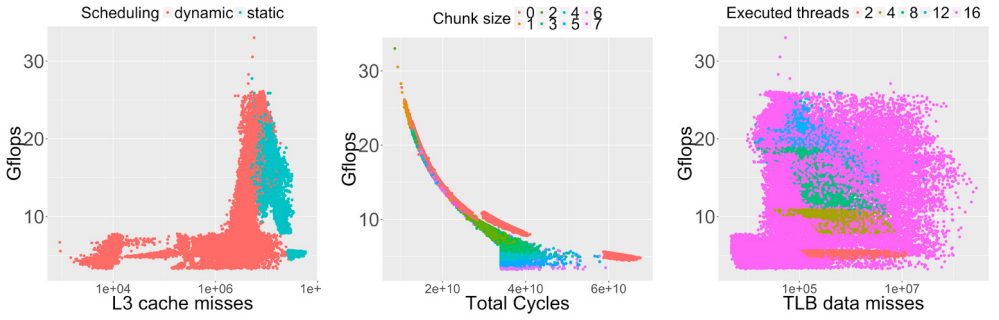
Table 1: Input vector for each algorithm.

Naive	Blocking
Number of threads	Number of threads
Scheduling policy	Scheduling policy
Chunk size	Chunk size
	Block size X
	Block size Y

1. **Input Vector**: It is defined by the parameters listed in Table 1. We considered different parameters available in OpenMP such as the number of threads, the loop scheduling policy (static or dynamic), and the chunk size (which defines how many loop iterations will be assigned to each thread at a time). Moreover, we considered the size blocks in the X and Y domains in the blocking implementation. The vertical direction is preserved in order to maximize the efficiency of prefetching mechanisms.
2. **Hardware Counters Vector:** We used the PAPI library to collect information from hardware counters. We considered the following metrics as the most relevant ones: L3 total cache misses (**PAPI_L3_TCM event**), data translation lookaside buffer misses (**PAPI_TLB_DM event**), and total cycles (**PAPI_TOT_CYC event**).
3. **Performance Vector:** The output vector, which uses billions of floating-point operations per second (GFLOPS) and execution time as performance characterization metrics.

3.2 Hardware Counters Behavior

Figure 1 illustrates how the performance of the 7-point Jacobi kernel is affected by the input variables and their relations with hardware counters. Each point represents one experiment. For instance, Figure 1(a) shows that the scheduling policy creates two separated sets when GFLOPS values are related with L3 cache misses. The same behavior is observed in Figure 1(b) for the chunk size when GFLOPS values are observed with respect to total number of cycles. The situation is rather different when GFLOPS are observed with respect to the amount of TLB data misses.



(a) L3 cache misses when varying the scheduling policy.

(b) Total cycles when varying the chunk size.

(c) TLB data misses when varying the number of threads.

Figure 1: Hardware counters when running the 7-point Jacobi (blocking implementation) on the Node 1 platform. This platform is detailed in Section 4.1.

3.3 Machine Learning Model

The proposed ML model is based on SVM, which is a supervised ML approach introduced in [4] and extended to regression problems where support vectors are represented by kernel functions [7]. The main idea of SVMs is to expand hyperplanes through the output vector. It has been employed to classify non-linear problems with non-separable training data by a linear decision surface (i.e. hardware counters behavior in previous section).

Our ML model was built on top of three consecutive layers, where output values of a layer are used as input values of the next layer (Figure 2). The input layer contains the configuration values from the input vector. The hidden layer contains three SVMs that take values from the input vector to simulate the behavior of hardware counters presented in the previous section. Finally, the output layer contains one SVM that takes each simulated value from the hidden layer to obtain the corresponding GFLOPS and execution time values (the latter are derived from an exponential fitting).

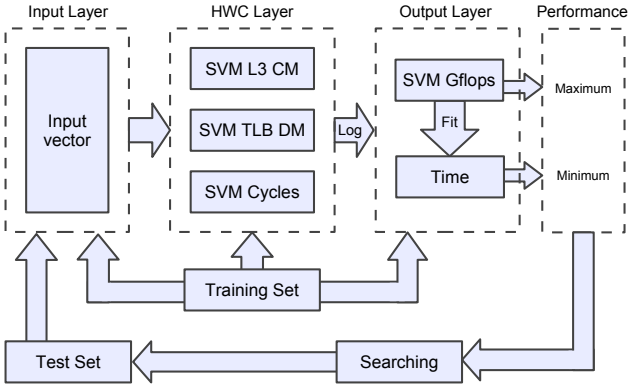


Figure 2: Flowchart of the proposed ML-based model.

4 Experiments

In this section we describe our experimental testbed and present the data analysis and results.

4.1 Experimental Testbed

We used two multi-core platforms to carry out the experiments. Their hardware configurations are shown in Table 2.

Table 2: Experimental testbed configurations.

	Node 1	**Node 2**
Processor	Xeon E5-2650	Xeon E5-4650
Clock (GHz)	2.0	2.7
Cores	8	8
Sockets	2	4
Threads	16	32
L3 cache size (MB)	20	20

Based on these platforms, Table 3 details all the configurations available for our optimization categories. As it can be noted, a brute force approach would be unfeasible, requiring more than 2 million simulations for the blocking implementation on Node 1 and more than 3 million simulations on Node 2.

Table 3: Optimizations set.

Optimization	**Parameters**	**Total configurations**	
		Node 1	**Node 2**
Number of threads	1	8	12
Scheduling policy	1	2	2
Chunk size	1	32	32
Block size X	1	64	64
Block size Y	1	64	64
Total for Naive	**3**	**512**	**768**
Total for Blocking	**5**	**2,097,152**	**3,145,728**

4.1.1 Training and validation sets

We created a training set by randomly selecting a subset from the configuration set presented in Table 3. Then, for each experiment we measured the hardware counters (L3 cache misses, data translation lookaside buffer misses and total cycles) and performance values (GFLOPS and execution time). Because hardware counters have very large values it was necessary to perform a dynamic range compression (log transformation) between the hidden layer and the output layer, as shown in Figure 2.

A random testing set was used since all SVMs in both the hidden and the output layers are trained to calculate new GFLOPS and execution time values through simulation. After that, we measured the accuracy of the model using statistical estimators. Finally, the maximum value of GFLOPS and the minimum value of execution time were selected and matched with their input values. Simulated and real values are compared to determine if the best performance obtained from the simulation is the same as the real best performance. Table 4 presents the total number of experiments that were performed to obtain the training and validation sets.

Table 4: Number of experiments in training and testing sets.

		Naive		**Blocking**	
Stencil	**Set**	**Node 1**	**Node 2**	**Node 1**	**Node 2**
7-point Jacobi	Training	44	38	2,355	4,054
	Testing	11	10	589	1,014
	Total	*55*	*48*	*44,794*	*49,152*
Seismic Wave	Training	211	237	2,176	371
	Testing	53	60	544	93
	Total	*264*	*297*	*6,849*	*1,020*

4.2 Analysis of Variance

In order to refine the results, we applied the ANOVA statistical model to analyze the influence of the different populations. We assume different populations for each variable and we assume that all populations have equal mean. Thus, we compute the statistical significance (*p-value*) to determine whether the hypothesis must be rejected or not: if this value is lower than 0.05 then the hypothesis is rejected and populations have different means. This analysis was divided in two

classical ANOVA models: one-way ANOVA, when only one factor affects all populations; and two-way ANOVA, when two factors affect all populations. In this analysis we used GFLOPS, L3 cache misses, TLB data misses and the number of cycles as population variables. Factors are defined by all values in the input vector (scheduling policy, chunk size and the number of threads). The results of *p-value* for the naive implementation are presented in Table 5.

As it can be noted, all factors rejected the hypothesis for the 7-point Jacobi, since *p-value* is lower than 0.05. All population variables have significant differences. For the seismic wave kernel, the hypothesis cannot be rejected for scheduling and chunk size variables. Table 6 shows results of two-way ANOVA to determine if combined variables affect populations. Results show that combining scheduling and chunk size with executed and available threads rejects the hypothesis, and variables have statistical differences if two factors are combined. So, the analysis of variance introduces the first assumption: factors produce statistical significance into selected variables.

Table 5: *p-value* of one-way ANOVA for the naive implementation.

	7-point Jacobi	Seismic Wave
Scheduling policy	2.58e-16	0.5284
Chunk size	1.37e-12	0.9985
Num. of threads	<2.2e-16	<2.2e-16
Num. of cores	<2.2e-16	<2.2e-16

Table 6: *p-value* of two-way ANOVA for the Seismic Wave kernel.

	p-value
Scheduling policy:Num. of threads	<2.2e-16
Scheduling policy:Num. of cores	0.4664
Chunk:Num. of threads	<2.2e-16
Chunk:Num. of cores	<2.2e-16

4.3 Results

We first evaluate the model with two statistical estimators: root mean square error (RMSE) and the coefficient of determination (R-square). The former represents the standard deviation of the differences between predicted values and real values whereas the latter represents how close the regression approximates the real data (R-square equal to 1 indicates a perfect fit of data regression). As it can be noted in Table 7, the regression model is highly accurate. For the naive implementation, the model presented an accuracy of up to 99.70% and 99.87% for GFLOPS and execution time metrics, respectively. For the blocking implementation, on the other hand, the model presented an accuracy of up to 98.22% and 99.71% for GFLOPS and execution time metrics, respectively.

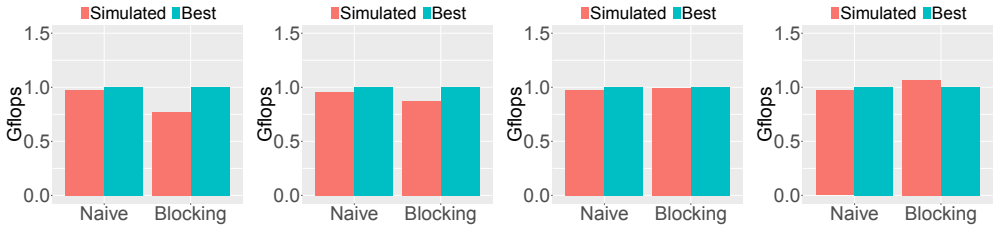
Table 7: RMSE and R-square for simulated values of the 7-point Jacobi and the Seismic Wave kernels.

			Naive		**Blocking**	
			Node 1	**Node 2**	**Node 1**	**Node 2**
7-point Jacobi	RMSE	GFLOPS	0.7941	1.0179	1.4185	1.6065
		Time	0.6642	2.5089	2.2537	3.4211
	R-square	GFLOPS	0.9782	0.9313	0.9627	0.8540
		Time	0.9879	0.8689	0.8881	0.8049
Seismic Wave	RMSE	GFLOPS	0.2273	0.6351	0.3158	0.4597
		Time	13.5391	212.282	15.6548	347.4940
	R-square	GFLOPS	0.9970	0.8334	0.9822	0.7313
		Time	0.9987	0.6263	0.9971	0.7494

Second, since the goal is to obtain the best performance, we compared the model with the best performance measurement from all data. Figure 3 present the results of this comparison. Red bars represent the normalized output of simulated best performance from the ML-based

model (maximum GFLOPS) whereas blue bars represent the normalized best performance from real data. Perfect fit for best performance is obtained when the simulated values are the same (or very close) to best performance values. This means that the predicted best performance actually matches the best performance from all data. The best performance for blocking implementation is described in [8].

Figures 3(a) and 3(b) compare the performance of simulated and real data for the 7-point Jacobi stencil. As it can be noted, the simulated performance achieved by the model for the naive implementation is very close to the best performance on both platforms. Simulated values for the blocking implementation, on the other hand, did not exactly reach the same best performance, although they are close to the real ones. For the seismic wave kernel, Figures 3(c) and 3(d) show that the simulated performance is very close to the best performance. The approximation of the best performance for 7-point Jacobi was 97.46% and 95.61% for the naive implementation on Node 1 and Node 2, respectively, whereas it was 76.79% and 87.01% for the blocking implementation on Node 1 and Node 2, respectively. For the seismic wave kernel, the approximation was 97.78% and 97.04% for the naive implementation on Node 1 and Node 2, respectively, whereas it was 99.62% and 101.06% for the blocking implementation on Node 1 and Node 2, respectively. Simulation for naive is easier than blocking because the model has less parameters to be considered. Differences between Node 1 and Node 2 show the impact of NUMA effects, which are difficult to simulate.



(a) GFLOPS for 7-point Jacobi: Node 1.
(b) GFLOPS for 7-point Jacobi: Node 2.
(c) GFLOPS for Seismic Wave: Node 1.
(d) GFLOPS for Seismic Wave: Node 2.

Figure 3: Normalized performance comparison between results from the ML-algorithm and results from the best performance experimentations.

5 Related Works

Some previous works proposed performance optimizations for stencil applications on heterogeneous architectures. In [12] the authors analyzed the performance of task scheduling algorithms. They concluded that different scheduling policies combined with different task sizes may considerably affect the efficiency and performance of seismic wave kernels. Similarly, in [5] the authors used a methodology to optimize stencil computations for multiple architectures (multi-core and accelerators). They worked on target cache reuse methodologies across single and multiple stencil sweeps, examining cache-aware algorithms as well as cache-oblivious techniques. Their results demonstrated that recent trends in memory system organization have reduced the efficacy of traditional cache-blocking optimizations. Analogously, in [10] the authors presented a stencil auto-tuning framework for multi-core architectures that converts a sequential stencil expression into tuned parallel implementations. Overall, the main problem of these works is

that the search domain can be very large and searching for the best configuration would take too much time.

Other works investigated the accuracy of regression models and ML algorithms in different contexts. In [14] the authors compared ML algorithms for characterizing the shared L2 cache behavior of programs on multi-core processors. The results showed that regression models trained on a given L2 cache architecture are reasonably transferable to other L2 cache architectures. In [19] the authors proposed a dynamic scheduling policy based on a regression model that is capable of responding to the changing behaviors of threads during execution.

Finally, in [18] the authors presented ML-based predictors to map parallelism to multi-cores. They considered several different parameters such as the number of threads and scheduling policies in OpenMP programs. In [9] the authors applied ML techniques to explore stencil configurations (code transformations, compiler flags, architectural features and optimization parameters). Their approach is able to select a suitable configuration that gives the best execution time and energy consumption. In [6] the authors improved performance of stencil computations by using a model based on cache misses.

6 Conclusions and Future Work

We proposed in this paper an ML-based model to simulate performance behavior of stencil computations on multi-core architectures. We showed that performance of two well-studied stencil kernels (7-point Jacobi and seismic wave) can be predicted with a high accuracy using hardware counters and the best configuration can be obtained from hardware counters and performance measurements. Our future works can be summarized in the following lines.

First, we believe that our model can be integrated into an auto-tuning framework to find the best performance configuration for a given stencil kernel. One possibility would be to use the Boast automatic source-to-source transformations framework [16]. Second, we expect to extend our methodology in order to capture complex behaviors on advanced architectures (NUMA effects, manycores, vectorization, spacetime blocking). Finally, we intend to design a model based on unsupervised ML algorithms to further improve our results.

7 Acknowledgments

This work has been granted by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS), and Bureau de Recherches Géologiques et Minières (Institut Carnot BRGM). Research has received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, grant agreement n° 689772.

References

[1] F. Z. Boito, R. V. Kassick, P. O. A. Navaux, and Y. Denneulin. Automatic I/O scheduling algorithm selection for parallel file systems. *Concurrency and Computation: Practice and Experience*, 28(8):2457–2472, 2016. cpe.3606.

[2] M. Castro, L. F. W. Góes, and J.-F. Méhaut. Adaptive thread mapping strategies for transactional memory applications. *Journal of Parallel and Distributed Computing*, 74(9):2845 – 2859, 2014.

[3] M. Christen, O. Schenk, and H. Burkhart. Automatic code generation and tuning for stencil kernels on modern shared memory architectures. *Comput. Sci.*, 26(3-4):205–210, June 2011.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[5] K. Datta, S. Kamil, S. Williams, L. Oliker, J. Shalf, and K. Yelick. Optimization and performance modeling of stencil computations on modern microprocessors. *SIAM Rev.*, 51(1):129–159, February 2009.

[6] R. de la Cruz and M. Araya-Polo. *Modeling Stencil Computations on Modern HPC Architectures*, pages 149–171. Springer International Publishing, Cham, 2015.

[7] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997.

[8] F. Dupros, F. Boulahya, H. Aochi, and P. Thierry. Communication-avoiding seismic numerical kernels on multicore processors. In *International Conference on High Performance Computing and Communications (HPCC)*, pages 330–335, Aug 2015.

[9] A. Ganapathi, K. Datta, A. Fox, and D. Patterson. A case for machine learning to optimize multicore performance. In *Proceedings of the First USENIX Conference on Hot Topics in Parallelism*, HotPar'09, pages 1–1, Berkeley, CA, USA, 2009. USENIX Association.

[10] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams. An auto-tuning framework for parallel multicore stencil computations. In *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–12, April 2010.

[11] T. M. Malas, G. Hager, H. Ltaief, H. Stengel, G. Wellein, and D. E. Keyes. Multicore-optimized wavefront diamond blocking for optimizing stencil updates. *SIAM J. Scientific Computing*, 37(4), 2015.

[12] V. Martínez, D. Michéa, F. Dupros, O. Aumage, S. Thibault, H. Aochi, and P. O. A. Navaux. Towards seismic wave modeling on heterogeneous many-core architectures using task-based runtime system. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2015 27th International Symposium on*, pages 1–8, Oct 2015.

[13] R. Mijakovic, M. Firbach, and M. Gerndt. An architecture for flexible auto-tuning: The periscope tuning framework 2.0. In *International Conference on Green High Performance Computing (ICGHPC)*, pages 1–9, Feb 2016.

[14] J. K. Rai, A. Negi, R. Wankar, and K. D. Nayak. On prediction accuracy of machine learning algorithms for characterizing shared l2 cache behavior of programs on multicore processors. In *International Conference on Computational Intelligence, Communication Systems and Networks (CICSYN)*, pages 213–219, July 2009.

[15] Y. Tang, R. A. Chowdhury, B. C. Kuszmaul, C.-K. Luk, and C. E. Leiserson. The pochoir stencil compiler. In *ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 117–128, New York, NY, USA, 2011. ACM.

[16] B. Videau, V. Marangozova-Martin, and J. Cronsioe. BOAST: Bringing Optimization through Automatic Source-to-Source Transformations. In *International Symposium on Embedded Multicore/Manycore System-on-Chip (MCSoC)*, Tokyo, Japan, 2013. IEEE Computer Society.

[17] D. Vladuic, A. Cernivec, and B. Slivnik. Improving job scheduling in grid environments with use of simple machine learning methods. In *International Conference on Information Technology: New Generations*, pages 177–182, April 2009.

[18] Z. Wang and M. F. P. O'Boyle. Mapping parallelism to multi-cores: A machine learning based approach. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '09, pages 75–84, New York, NY, USA, 2009. ACM.

[19] L. Weng, C. Liu, and J.-L. Gaudiot. Scheduling optimization in multicore multithreaded microprocessors through dynamic modeling. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, pages 5:1–5:10, New York, NY, USA, 2013. ACM.