# Performance Evaluation of Stencil Computations based on Source-to-Source Transformations

Víctor Martínez[1], Matheus S. Serpa[1], Pablo J. Pavan[1], Edson L. Padoin[2], and Philippe O. A. Navaux[1]

[1] Informatics Institute, UFRGS, Brazil
{victor.martinez,msserpa,pablo.pavan,navaux}@inf.ufrgs.br
[2] Department of Exact Sciences and Engineering, UNIJUI, Brazil
padoin@unijui.edu.br

**Abstract.** Stencil computations are commons in High Performance Computing (HPC) applications, they consist in a pattern that replicates the same calculation in a data domain. The Finite-Difference Method is an example of stencil computations and it is used to solve real problems in diverse areas related to Partial Differential Equations (electromagnetics, fluid dynamics, geophysics, etc.). Although a large body of literature on optimization of this class of applications is available, the performance evaluation and its optimization on different HPC architectures remain a challenge. In this work, we implemented the 7-point Jacobian stencil in a Source-to-Source Transformation Framework (BOAST) to evaluate the performance of different HPC architectures. Achieved results present that the same source code can be executed on current architectures with a performance improvement, and it helps the programmer to develop the applications without dependence on hardware features.

**Keywords:** Stencil applications · Heterogeneous architectures · Source-to-source transformation · Performance evaluation · Performance improvement.

## 1 Introduction

The trend of High Performance Computing (HPC) applications is to exploit all the processing power of multicore and heterogeneous architectures. Currently, there are several architectural features and programming models to be considered when applications are developed. This produces a complex situation of many interdependent factors, at software and hardware levels, that may severely influence the application performance (non-uniform memory access, vectorization, compiler optimizations, memory policies, communications, etc.) [2, 16, 22].

Stencil-based computations are an example of HPC applications, they are defined by a pattern that replicates the same calculation in all the data domain. For instance, the Finite-Difference Method (FDM) to discretize the Partial Differential Equations (PDE) 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 3D Cartesian grid. The algorithm then moves to the next point applying the same computation to complete the entire spatial grid. The number of points in each direction depends on the order of the approximation. From the numerical analysis point of view, the FDM is the basis of a significant fraction of numerical solvers in many fields (i.e., electromagnetics, fluid dynamics or geophysics) [1, 5, 8, 17].

A large body of literature on stencil optimization is available, but the performance evaluation remains a challenge on current architectures [3, 7, 15, 23]. Most of this methods are limited by architectural issues. In this work, we describe the procedure to evaluate the performance and to optimize the stencil computations on HPC architectures by using a framework for Source-to-source (S2S) transformations. We used a 7-point Jacobi stencil implemented on BOAST [4]. The main advantage of this framework is that applications can be executed in different HPC architectures without changing the source code.

This paper is organized as follows: Section 2 provides the fundamentals of the stencil under study; Section 3 explains the framework used to develop the application; Section 4 describes the methodology, the experiments, the testbed, and the achieved results; Section 5 describes the related work; and finally, Section 6 concludes this paper.

## 2 Stencil Model

In this section, we present the numerical model. The 7-point Jacobi stencil is a reference example of a numerical kernel used in various context in order to evaluate the impact of advanced reformulation or the impact of the underlying architecture. Known to be severely memory-bound, this kernel can be described as a proxy of complex stencils like those corresponding to geophysical applications. The numerical review can be found in [6].

This stencil model also corresponds to the standard discretization of the elliptic 3D Heat equation (1) [18]. Due to its simplicity, the Finite-Differences Method (FDM) is widely used to solve this numerical model, when discretizing Partial Differential Equations (PDE). From the numerical analysis point of view, the FDM computational procedure consists in using the neighboring points in horizontal, vertical or diagonal directions to calculate the current point.

$$\begin{aligned} B_{i,j,k} =& \alpha A_{i,j,k} \\ &+ \beta(A_{i-1,j,k} + A_{i,j-1,k} + A_{i,j,k-1} + A_{i+1,j,k} + A_{i,j+1,k} + A_{i,j,k+1}) \end{aligned} \tag{1}$$

Calculation of this numerical equation needs seven values, one from current point plus six from neighbor points (one previous and one next on each 3D axes). Representation of stencil size is presented in Figure 1.

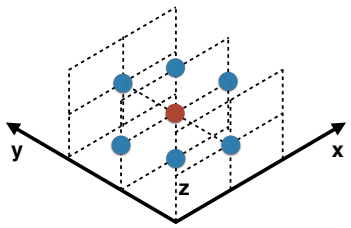


Fig. 1: Size of 7-point Jacobi stencil.

A standard metric available to characterize a stencil kernel is the Arithmetic Intensity (AI) that can be defined as the ratio between the floating point operations and the memory transfers. In the case of the 7-point Jacobi kernel, the lower-bound of the arithmetic intensity is 0.18 [14]. The synthetic pseudo-code of this kernel could be found in Algorithm 1.

**Algorithm 1:** Pseudo-code for the 7-point Jacobi stencil.

**for** $i = 1$ to $N_x$ **do**
  **for** $j = 1$ to $N_y$ **do**
    **for** $k = 1$ to $N_z$ **do**

$$X^{n+1}(i,j,k) = X^n(i,j,k) + X^n(i,j,k+1) + X^n(i,j,k-1) + X^n(i,j+1,k) + X^n(i,j-1,k) + X^n(i+1,j,k) + X^n(i-1,j,k)$$

    **end for**
  **end for**
**end for**

## 3 S2S Frameworks

In this section, we present the S2S transformation framework. This transformation procedure has been used to improve the performance of HPC applications. Some considerations of S2S transformations are:

i) It is not applied in random order,
ii) it could not cause an improvement on the program,
iii) sometimes this transformation depends on the target machine.

The procedure to transform the source code is applied by finding some pattern in the program, then perform a set of replacements defined by a set of rules. Some conventional rules are simplifications of constant computation, loop unrolling or loop elimination [13].

Automatic parallelization methods have been successfully applied to improve the performance. This auto-parallelization method can integrate data-

dependence profiling, task parallelism extraction and source-to-source transformation. Based on program analysis tools, some parallelization approaches automatically generate parallel code without requiring programmers to indicate parallel code sections, by extracting coarse-grained task parallelism, to transform sequential source code to parallel code, which exploits both loop parallelism and task parallelism without special compiler support [26].

On heterogeneous architectures, application development requires a lot of effort and investment from the programmer. This problem will become more prominent when HPC architectures are frequently updated to keep with market trends. In these scenarios, automatic parallelization tools will definitely have an important role to play, they would be the ability to perform pertinent domain decomposition of the serial code to maximize utilization of the available computational elements [11]

In this work, we used an S2S framework called BOAST [4]. It provides programmers with a tool to develop computing kernels. The workflow of BOAST is defined by the following steps: 1) the developer starts from an application kernel, and writes it in a dedicated language (Ruby); 2) the S2S parameters define the output source code that will be generated (Sequential C, OpenMP, Fortran, OpenCL, CUDA); 3) The resulting code source is then built according to the specified compiler; 4) based on the results, other optimizations can be selected; 5) the resulting kernel is then added to the program [24].

## 4 Experimental Methodology

In this section, we present the experiments and the results of our approach. We used as data domain a three-dimensional Cartesian grid of size 512x512x512, and 190-time iterations, to execute the Jacobi stencil (the benchmark for our experiments).

### 4.1 Kernel definition

In order to define the kernel, we use the available language description: the keywords *decl* and *pr* defined. The *decl* method is used to declare variables or procedures and functions. The *pr* method calls the public *pr* method of objects it is called on. Each BOAST object is responsible for printing itself correctly depending on the BOAST configuration at the time the print public method is called.

BOAST defines several classes that are used to represent the structure of the code, these classes can be algebraic related and control flow related (i.e, OpenMP). On the other hand, the classical control structures were also implemented, *If*, *For* are abstractions in BOAST matching the behavior of corresponding control structures in other languages. The last control structure is *Procedure*. It describes procedures (Fortran), functions (sequential C or OpenMP), and kernels (CUDA). The kernel definition and the usage of keywords, classes and control structures can be found in the figure 2. We used this kernel definition to

be executed into two heterogeneous machines. The advantage of this approach is we don't change the source code. BOAST makes de S2S transformation and uses available compilers (icc, gcc, and nvcc) to execute the kernel.

```
def Stencil_Probe(omp = false, cuda = false)

  p_Stencil_Probe = Procedure("Stencil_Probe",
    [nx, ny, nz, tx, ty, tz, timesteps,
    v_prev, v_next, v_vel, v_coeff, $SIZE_STENCIL] ){
    decl i = Int("i")
    decl j = Int("j")
    decl k = Int("k")
    decl t = Int("t")
    decl wst = Int("wst")
    decl value = Real("value")
      f = For(t, 0, timesteps - 1){
        iter_kernel = For(i, $SIZE_STENCIL,
          nx - $SIZE_STENCIL - 1){
          pr For(j, $SIZE_STENCIL, ny - $SIZE_STENCIL - 1){
            pr For(k, $SIZE_STENCIL, nz - $SIZE_STENCIL - 1){
              it_prev = compute_prev(v_prev, v_next, v_vel,
                v_coeff)
              it_next = compute_next(v_prev, v_next, v_vel,
                v_coeff)
            }
          }
        }
        if omp
          pr OpenMP::ParallelFor( :schedule => "runtime" ) {
            pr it_prev
          }
          pr OpenMP::ParallelFor( :schedule => "runtime" ) {
            pr it_next
          }
        else
          pr iter_kernel
        end
      }
    pr f
  }
  kernel = CKernel::new
  kernel.procedure = p_Stencil_Probe
  pr p_Stencil_Probe
  return kernel
end
```

Fig. 2: Kernel definition in BOAST

### 4.2 Tesbed

We performed our experiments into two accelerator architectures, some details are presented in Table 1:

- The *KNL machine* provides an Intel Xeon Phi architecture (Knights Landing). The Knights Landing is the code name for the second-generation Intel Xeon Phi family. It is a Many Integrated Core (MIC) architecture that delivers massive thread parallelism, data parallelism, and memory bandwidth in a CPU form factor for high throughput workloads. It is a standard, standalone processor that can boot an off-the-shelf operating system [21].
- Tha *Blaise machine* provides a classical heterogeneous architecture. It is composed of two multicore processors (Intel Xeon) and four GPUs (NVIDIA Tesla P100). It uses the GPU devices as accelerators of kernel computing.

Table 1: HPC architectures Testbed

| | ***KNL*** | ***Blaise*** |
|---|---|---|
| **Standalone CPU** | Intel Xeon Phi 7250 | Intel Xeon E5-2699 (2x) |
| **Co-processor** | Intel Xeon Phi 7250 | NVIDIA Tesla P100 (4x) |
| **Total number of CPU threads** | 272 | 88 |
| **CPU compiler** | icc 18.0.1 | gcc 5.4.0 |

The two machines have their particular features, and classical application developing of these architectures requires different programming models (shared memory, or stream multiprocessing). In this sense, the S2S transformation can help to implement the stencil computations in an easy way.

### 4.3 Results

Since we developed the Jacobi stencil in the S2S framework, we used the BOAST runtime to execute our experiments. Experiments were executed 30 times, and we measured the average of executing time. A Shapiro-Wilk test for time measurement was performed to confirm normality.

For KNL machine, we called the S2S transformation to execute the kernel by sequential C and by OpenMP. As we can see in Figure 3, the BOAST runtime optimize the parallel execution and we obtain a performance improvement when we compare the parallel and sequential executions. In the same way, we executed sequential C, OpenMP, and CUDA for Blaise machine; as we present in Figure 3, the BOAST runtime optimizes the performance when multicore and accelerators are used.

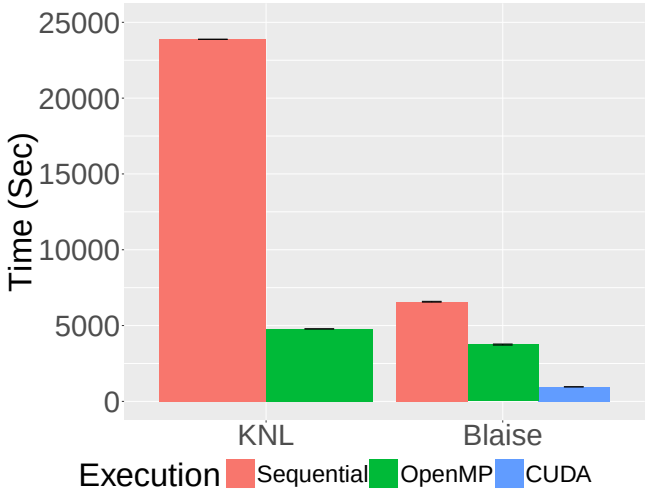


Fig. 3: Average time of stencil executions.

When we analyze the performance of stencil executions we found an improvement by minimizing the execution time; although the experiments seem don't reach a peak of best performance. As we noted by the speedup measure in Figure 4. In this context, if we want a better-optimized implementation we think that we need to improve the source code according to architectural features for each machine.

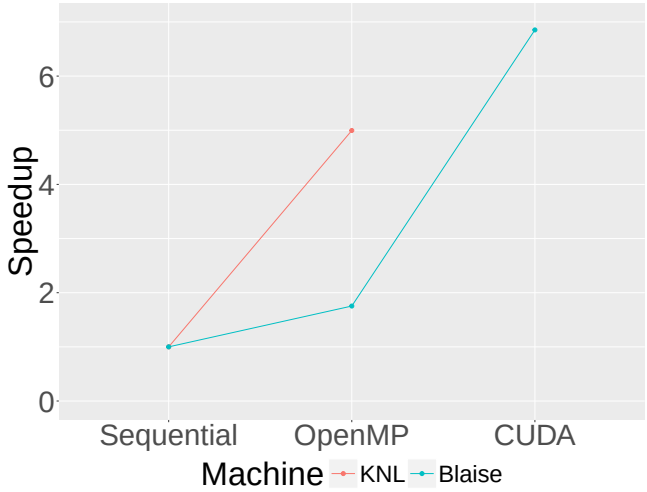


Fig. 4: Speedup of stencil executions.

## 5 Related Work

In this section, we present some related work. Our research is oriented to S2S transformations applied to Jacobian stencil, the main idea is to evaluate this application to transformations on several architectures and several programming languages. S2S transformations can be done in several ways, we focused on two of them: transformations on the same programming language, or transformations using a meta-language to generate code in different languages.

The Inject/J software transformation language is an example of the first group, as a dynamically typed scripting language for S2S transformations of Java programs [10]. In [20], the authors implement a framework to transform sequential code into OpenMP parallel code; they present the Checkpoint Aided Parallel Execution (CAPE) methodology to modify the sequential parts to be executed in parallel, and the Turing eXtended Language (TXL) is composed by a description of the structures to transform the programs and a set of transformation rules. In [25], the authors introduce an end-to-end framework for automatically transforming stencil-based CUDA programs to exploit inter-kernel data locality; this work formulated the GPU kernel fission/fusion problem and demonstrated effectiveness, the programmer can compile the new code using the CUDA compilers. In addition, another optimization is tuning of thread block size for kernels generated to achieve high occupancy. In [9], the authors presented the S2S transformation to Jacobian calculation of functions defined by Fortran code; they used a framework called ELIAD and implemented in Java, it uses Automatic Differentiation (AD) to create new code that calculates the numerical values of the variable and its derivatives with respect to the independent variables, it is a bi-directional data flow analysis to determine active variables from user-specified independent and dependent variables.

The second group of S2S transformation focuses on changes in the programming language. In [19], the authors propose a source-to-source compiler able to transform an OpenMP C code into a CUDA code, the generated code is fully NVIDIA CUDA compliant and can be compiled using the nvcc compiler. The entire transformation process includes starting from the pragma split-up and the kernel generation, passing through the data visibility clauses management and ending with the device memory management and the kernel launch system. In [12], the authors described a compiler framework for translating standard OpenMP shared-memory programs into CUDA-based GPU programs; they include a kernel region identifying algorithm, by applying OpenMP as a front-end programming model, the proposed translator could convert the loop-level parallelism of the OpenMP programming model into the data parallelism of the CUDA programming model in a natural way; they have also identified several key transformation techniques to enable efficient GPU global memory access: parallel loop-swap and matrix transpose techniques for regular applications, as the Jacobi stencil, and loop collapsing for irregular ones.

## 6 Conclusion and Future Work

In this research, we presented an implementation of 7-point Jacobi stencil based on S2S transformations and analyzed its performance on current HPC architectures. We used the BOAST framework to demonstrate that this approach can improve the application performance on different HPC machines. The S2S transformation provides the programmer an easy way to develop the HPC applications without concerning the hardware configuration. In contrast, we also confirmed that performance is dependent on the architecture hardware; as we presented, if we want to reach the best performance peak, we need to improve the source code according to the architectural features.

Our future work is focused on two perspectives: first, optimization of more complex stencils (i.e, geophysics stencils) by using the S2S transformations; second, we also believe that performance of S2S transformations can be improved by auto-tuning techniques based on Machine Learning algorithms

## Acknowledgments

This work has been granted by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)*, the *Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq)*, the *Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS)*. Research has received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the *HPC4E Project*, grant agreement n.º 689772. It was also supported by Intel under the Modern Code project, and the *PETROBRAS* oil company under Ref. 2016/00133-9. We also thank to *RICAP*, partially funded by the Ibero-American Program of Science and Technology for Development (*CYTED*), Ref. 517RT0529.

## References

1. Breuer, A., Heinecke, A., Bader, M.: Petascale local time stepping for the ADER-DG finite element method. In: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016. pp. 854–863 (2016)
2. Buchty, R., Heuveline, V., Karl, W., Weiss, J.P.: A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators. Concurrency and Computation: Practice and Experience **24**(7), 663–675 (2012). https://doi.org/10.1002/cpe.1904
3. Christen, M., Schenk, O., Burkhart, H.: Automatic code generation and tuning for stencil kernels on modern shared memory architectures. Comput. Sci. **26**(3-4), 205–210 (Jun 2011)
4. Cronsioe, J., Videau, B., Marangozova-Martin, V.: Boast: Bringing optimization through automatic source-to-source transformations. In: 2013 IEEE 7th International Symposium on Embedded Multicore Socs. pp. 129–134 (Sept 2013). https://doi.org/10.1109/MCSoC.2013.12

5. Datta, K., Kamil, S., Williams, S., Oliker, L., Shalf, J., Yelick, K.: Optimization and performance modeling of stencil computations on modern microprocessors. SIAM Rev. **51**(1), 129–159 (Feb 2009). https://doi.org/10.1137/070693199
6. Datta, K., Williams, S.W., Volkov, V., Carter, J., Oliker, L., Shalf, J., Yelick, K.: Auto-Tuning Stencil Computations on Multicore and Accelerators. CRC Press, Taylor & Francis Group (2010)
7. Dupros, F., Boulahya, F., Aochi, H., Thierry, P.: Communication-avoiding seismic numerical kernels on multicore processors. In: High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on. pp. 330–335 (Aug 2015). https://doi.org/10.1109/HPCC-CSS-ICESS.2015.230
8. Dupros, F., Do, H., Aochi, H.: On scalability issues of the elastodynamics equations on multicore platforms. In: Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013. pp. 1226–1234 (2013)
9. Forth, S.A., Tadjouddine, M., Pryce, J.D., Reid, J.K.: Jacobian code generated by source transformation and vertex elimination can be as efficient as hand-coding. ACM Trans. Math. Softw. **30**(3), 266–299 (Sep 2004). https://doi.org/10.1145/1024074.1024076, http://doi.acm.org/10.1145/1024074.1024076
10. Genssler, T., Kuttruff, V.: Source-to-source transformation in the large. In: Böszörményi, L., Schojer, P. (eds.) Modular Programming Languages. pp. 254–265. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
11. Khan, M., Priyanka, N., Ahmed, W., Radhika, N., Pavithra, M., Parimala, K.: Understanding source-to-source transformations for frequent porting of applications on changing cloud architectures. In: 2014 International Conference on Parallel, Distributed and Grid Computing. pp. 350–354 (Dec 2014). https://doi.org/10.1109/PDGC.2014.7030769
12. Lee, S., Min, S.J., Eigenmann, R.: Openmp to gpgpu: A compiler framework for automatic translation and optimization. SIGPLAN Not. **44**(4), 101–110 (Feb 2009). https://doi.org/10.1145/1594835.1504194, http://doi.acm.org/10.1145/1594835.1504194
13. Loveman, D.B.: Program improvement by source-to-source transformation. J. ACM **24**(1), 121–145 (Jan 1977). https://doi.org/10.1145/321992.322000, http://doi.acm.org/10.1145/321992.322000
14. Martnez, V., Dupros, F., Castro, M., Navaux, P.: Performance improvement of stencil computations for multi-core architectures based on machine learning. Procedia Computer Science **108**, 305 – 314 (2017). https://doi.org/https://doi.org/10.1016/j.procs.2017.05.164, http://www.sciencedirect.com/science/article/pii/S1877050917307408, international Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland
15. Mijakovic, R., Firbach, M., Gerndt, M.: An architecture for flexible auto-tuning: The periscope tuning framework 2.0. In: International Conference on Green High Performance Computing (ICGHPC). pp. 1–9 (Feb 2016). https://doi.org/10.1109/ICGHPC.2016.7508066
16. Mittal, S., Vetter, J.S.: A survey of cpu-gpu heterogeneous computing techniques. ACM Comput. Surv. **47**(4), 69:1–69:35 (Jul 2015). https://doi.org/10.1145/2788396

17. Moczo, P., Robertsson, J., Eisner, L.: The finite-difference time-domain method for modeling of seismic wave propagation. In: Advances in Wave Propagation in Heterogeneous Media, Advances in Geophysics, vol. 48, chap. 8, pp. 421–516. Elsevier - Academic Press (2007)
18. Nguyen, A., Satish, N., Chhugani, J., Kim, C., Dubey, P.: 3.5-d blocking optimization for stencil computations on modern cpus and gpus. In: 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–13 (Nov 2010). https://doi.org/10.1109/SC.2010.2
19. Noaje, G., Jaillet, C., Krajecki, M.: Source-to-source code translator: Openmp c to cuda. In: 2011 IEEE International Conference on High Performance Computing and Communications. pp. 512–519 (Sept 2011). https://doi.org/10.1109/HPCC.2011.73
20. Renault, E., Ancelin, C., Jimenez, W., Botero, O.: Using source-to-source transformation tools to provide distributed parallel applications from openmp source code. In: 2008 International Symposium on Parallel and Distributed Computing. pp. 197–204 (July 2008). https://doi.org/10.1109/ISPDC.2008.65
21. Sodani, A., Gramunt, R., Corbal, J., Kim, H.S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., Liu, Y.C.: Knights landing: Second-generation intel xeon phi product. IEEE Micro **36**(2), 34–46 (Mar 2016). https://doi.org/10.1109/MM.2016.25
22. Stojanovic, S., Bojic, D., Bojovic, M., Valero, M., Milutinovic, V.: An overview of selected hybrid and reconfigurable architectures. In: Industrial Technology (ICIT), 2012 IEEE International Conference on. pp. 444–449 (March 2012). https://doi.org/10.1109/ICIT.2012.6209978
23. Tang, Y., Chowdhury, R.A., Kuszmaul, B.C., Luk, C.K., Leiserson, C.E.: The pochoir stencil compiler. In: ACM Symposium on Parallelism in Algorithms and Architectures. pp. 117–128. SPAA '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/1989493.1989508, http://doi.acm.org/10.1145/1989493.1989508
24. Videau, B., Pouget, K., Genovese, L., Deutsch, T., Komatitsch, D., Desprez, F., Mhaut, J.F.: Boast: A metaprogramming framework to produce portable and efficient computing kernels for hpc applications. The International Journal of High Performance Computing Applications **32**(1), 28–44 (2018). https://doi.org/10.1177/1094342017718068, https://doi.org/10.1177/1094342017718068
25. Wahib, M., Maruyama, N.: Automated gpu kernel transformations in large-scale production stencil applications. In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing. pp. 259–270. HPDC '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2749246.2749255, http://doi.acm.org/10.1145/2749246.2749255
26. Zhao, B., Li, Z., Jannesari, A., Wolf, F., Wu, W.: Dependence-based code transformation for coarse-grained parallelism. In: Proceedings of the 2015 International Workshop on Code Optimisation for Multi and Many Cores. pp. 1:1–1:10. COSMIC '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2723772.2723777, http://doi.acm.org/10.1145/2723772.2723777