# Introducing the *Semi-stencil* Algorithm

Raúl de la Cruz, Mauricio Araya-Polo, and José María Cela

Barcelona Supercomputing Center,
29 Jordi Girona, 08034 Barcelona, Spain
{raul.delacruz,mauricio.araya}@bsc.es

**Abstract.** Finite Difference (FD) is a widely used method to solve Partial Differential Equations (PDE). PDEs are the core of many simulations in different scientific fields, e.g. geophysics, astrophysics, etc. The typical FD solver performs stencil computations for the entire 3D domain, thus solving the differential operator. This computation consists on accumulating the contribution of the neighbor points along the cartesian axis. It is performance-bound by two main problems: the memory access pattern and the inefficient re-utilization of the data. We propose a novel algorithm, named "semi-stencil", that tackle those two problems. Our first target architecture for testing is Cell/B.E., where the implementation reaches 12.4 GFlops (49% peak performance) per SPE, while the classical stencil computation only reaches 34%. Further, we successfully apply this code optimization to an industrial-strength application (Reverse-Time Migration). These results show that semi-stencil is useful stencil computation optimization.

**Keywords:** Stencil computation, Reverse-Time Migration, High Performance Computing, Cell/B.E., heterogeneous multi-core.

## 1 Introduction

Astrophysics [1], Geophysics [2], Quantum Chemistry [3] and Oceanography [4] are examples of scientific fields where large computer simulations are frequently deployed. These simulations have something in common, the mathematical models are represented by Partial Differential Equations (PDE), which are mainly solved by the Finite Difference (FD) method. Large simulations may consume days of supercomputer time, if they are PDE+FD based, most of this execution time is spent in stencil computation. For instance, Reverse-Time Migration (RTM) is a seismic imaging technique in geophysics, of the RTM kernel execution time up to 80% [5] is spent in the stencil computation.

In this paragraph, we first review the stencil computation characteristics, then we identify its main problems. Basically, the stencil central point accumulates the contribution of neighbor points in every axis of the cartesian system. This operation is repeated for every point in the computational domain, thus solving the spatial differential operator, which is the most computational expensive segment of a PDE+FD solver. We identify two main problems:

– first, the non-contiguous memory access pattern. In order to compute the central point of the stencil, a set of neighbors have to be accessed, some of these neighbor points are far in the memory hierarchy, thus paying many cycles in latencies. Furthermore, depending on the stencil order (associated to the number of neighbors that contribute in the spatial differential operator) this problem becomes even more important, due to that the required data points are even more expensive to access.
– Second, the low computation/access and re-utilization ratios. After gather the set of data points just one central point is computed, and only some of those accessed data points will be useful for the computation of the next central point.

We introduce the *semi-stencil* algorithm, which improves the stencil computation tackling the above mentioned problems. We remark that this algorithm changes the structure of the stencil computation, but anyway it can be generally applied to most of the stencil based problems. The semi-stencil algorithm computes half the contributions required by a central point, but for two central points at the same time, with one a stencil computation is completed, and the other half central point pre-computes the stencil computation of another point. This algorithm reduces the number of neighboring points loaded per computation, this is achieved by just accessing the points required to compute half the stencil. This helps with the first problem of the stencil computation. At every step of the algorithm half of the stencil for the next step is pre-computed. The number of floating point operations remain the same, but because the load number is reduced the computation-access ratio is higher. So, tackling the second problem of the stencil computation.

Currently, the GHz race for higher frequencies has slowed down due to technological issues. Therefore, the main source of performance comes from the exploitation of multi-core architectures. Among such architectures, the Cell/B.E. (our target achitecture) exhibit appealing features as energy efficiency and remarkable computing power, which is demanded by the applications which rely on stencil computations.

The remainder of this paper is organized as follows: Section 2 introduces the classical stencil algorithm and its problems. In Section 3 we review the principal techniques that help to cope with the stencil problem. Section 4 introduces the novel semi-stencil algorithm, its internals, features and considerations. In Section 5, we evaluate the performance. Finally, Section 6 presents our conclusions.

## 2 The Stencil Problem

The stencil computes the spatial differential operator, which is required to solve a PDE+FD. A multidimensional grid (often a huge 3D data structure) is traversed, where the elements are updated with weighted contributions. Figure 1.b) depicts classical generic stencil structure. The $\ell$ parameter represents the number of neighbors to be used at each direction of the cartesian axis.

Two major problems can be inferred from this computation, the first one is the sparse memory access pattern [6,7]. Data is stored in $Z$-major form, therefore accesses across the other two axis ($X$ and $Y$) may be significantly more expensive (see Figure 1(a)) latency-wise. The $\ell$ parameter has a direct impact on this problem, the larger the $\ell$ value the more neighbors of each axis have to be loaded to compute $X^t_{i,j,k}$. The $\ell$ points of the $Z$ axis are stored sequentially in memory, and the cost to fetch them is low.

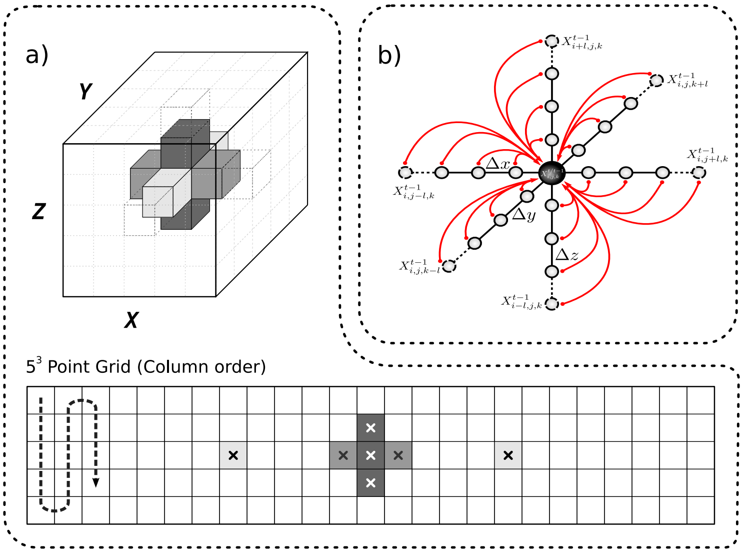

**Fig. 1.** (a) The memory access pattern for a 7-point stencil. The sparsity of the required data to compute the stencil is higher in the last axis. (b) The generic stencil structure.

The second problem has two faces: the low floating-point instruction to memory access ratio, and the poor reuse of the accessed data [8]. We state a simple metric to expose these two faces: **FP/Mem**. This ratio is the result of dividing the number of floating-points instructions (Multiply-Adds) per loads and stores during one $X^t$ computation step.

$$\begin{aligned} FP/Mem_{Classical} &= \frac{FloatingPoint\ Instructions}{Memory\ Accesses} = \frac{MultiplyAdd\ Instructions}{X^{t-1}\ Loads + X^t\ Stores} \\ &= \frac{2 * dim * \ell + 1}{(2 * (dim - 1) * \ell + 1) + (1)} = \frac{2 * dim * \ell + 1}{2 * dim * \ell - 2 * \ell + 2} \end{aligned} \tag{1}$$

Equation 1 states this metric for the classical stencil. *dim* is the number of dimensions of our PDE, where for each dimension $2 * \ell$ Multiply-Add instructions are required. Also, one extra Multiply-Add instruction must be considered for self-contribution ($X^{t-1}_{i,j,k}$). The number of loads needed to compute the stencil change depending on the axis. Those dimensions that are not stride direction ($X$ and $Y$) require $2 * \ell$ loads each step iteration. On the other hand, the stride direction requires only 1 load because the remaining loads can be reused from previous iterations. Finally, 1 store for saving the result ($X^{t}_{i,j,k}$) is needed.

FP/Mem ratio depends on variables $dim$ and $\ell$, taking into account that $\ell$ is the only variable that may grow, the ratio tends to 1.5, which is very poor. This result along with previous research [9,10,8] shows that stencil computation is usually memory-bound. In other words, it is not possible to feed the computation with enough data to keep a high arithmetic throughput, thus the performance of this kind of algorithm is limited.

These concerns force us to pay special attention on how data is accessed. It is crucial to improve the memory access pattern (reducing the overall amount of data transfers), and to exploit the memory hierarchy as much as possible (reducing the overall transfer latency). Next section review the main approaches to the problem found in the literature.

## 3 State of the Art

Most of the contributions in the stencil computations field can be divided into three dissimilar groups: space blocking, time blocking and pipeline optimizations. The first two are related with blocking strategies widely used in cache architectures, while the last one is used to improve the performance at the pipeline level.

### Space Blocking

Space blocking algorithms try to minimize the access pattern problem of stencil computations. Space blocking is specially useful when the data structure does not fit in the memory hierarchy. The most representative algorithms of this kind are: tiling or blocking [9] and circular queue [11].

### Time Blocking

Time blocking algorithms perform loop unrolling over the time steps to exploit as much as possible a data already read, therefore increasing the data reuse. Time blocking may be used where there is no other computation between stencil sweeps such as: boundary condition, communication, I/O, where a time dependency exists. Optimizations of the time blocking can be divided in explicitly and implicitly algorithms: time-skewing [10] and cache-oblivious [8].

### Pipeline Optimizations

Low level optimizations include well-known techniques suitable to loops, like unrolling, fission, fusion, prefetch and software pipelining [12,13]. All those techniques have been successfully used for a long time in many other computational fields to improve the processor throughput and decrease the CPI (Cycle per Instruction).

## 4 The Semi-stencil Algorithm

The semi-stencil algorithm changes in noticeable way the structure of the stencil computation as well as the memory access pattern of the stencil computation. This new computation structure (depicted in Figure 2) consist in two phases: **forward** and **backward**, which are described in detail in Section 4.1. Besides, head and tail computations are described in last part of the chapter. The semi-stencil tackles the problems described in Section 2 in the following ways:

- It improves data locality since less data is required on each axis per iteration. This may have an important benefit for hierarchy cache architectures, where the non-contiguous axes of the 3D domain are more expensive latency-wise.
- The second effect of this new memory access pattern is the reduction of the total amount of loads per inner loop iteration, but keeping the same number of floating-point operations. Thereby it increases the data re-utilization and the $FP/Mem$ ratio. We calculate this ratio for the semi-stencil as follows:

$$FP/Mem_{Semi} = \frac{MultiplyAdd\ Instructions}{X^{t-1}\ Loads + X^{t}\ Stores} = \frac{2 * dim * \ell + 1}{dim * \ell - \ell + dim + 2} \quad (2)$$

  In Equation 2, and regarding the data reuse in the stride dimension, the number of loads for $X^{t-1}$ have decreased substantially, almost by a factor of 2. Due to the reduced number of loads less cycles are needed to compute internal loop, also it has the benefit of using fewer registers per iteration. This gives a chance to perform more aggressive low level optimizations, and avoid instruction pipeline stalls.

As shown in Figure 2.Left, the semi-stencil algorithm updates two points per step by reusing $X^{t-1}$ loads, but at the same time doubles the number of stores needed per step of $X^{t}$. Depending on the architecture, this could be a problem and a source of performance loss. For instance, hierarchy cache architectures with write allocate policy. Nowadays, some hierarchy cache architectures implement cache-bypass techniques as a workaround for this problem. On the other hand, the scratchpad memory based (SPM) architectures, like Cell/B.E., are immune to this problem. Therefore, semi-stencil mapping on the Cell/B.E. architecture should not be affected by this issue.

It is worth pointing out that the semi-stencil algorithm can be applied to any axis of a 3D stencil computation. This means that it can be combined with the classical stencil algorithm. Moreover, this novel algorithm is independent of any other optimization technique, like blocking or pipeline optimizations. In fact, we can stack semi-stencil algorithm over any other known technique.

In the following sections we will elaborate on the semi-stencil two phases.

### 4.1 Forward and Backward Update

Forward update is the first contribution that a point receives at time-step $t$. In this phase, when step $i$ of the sweep direction axis is being computed, the

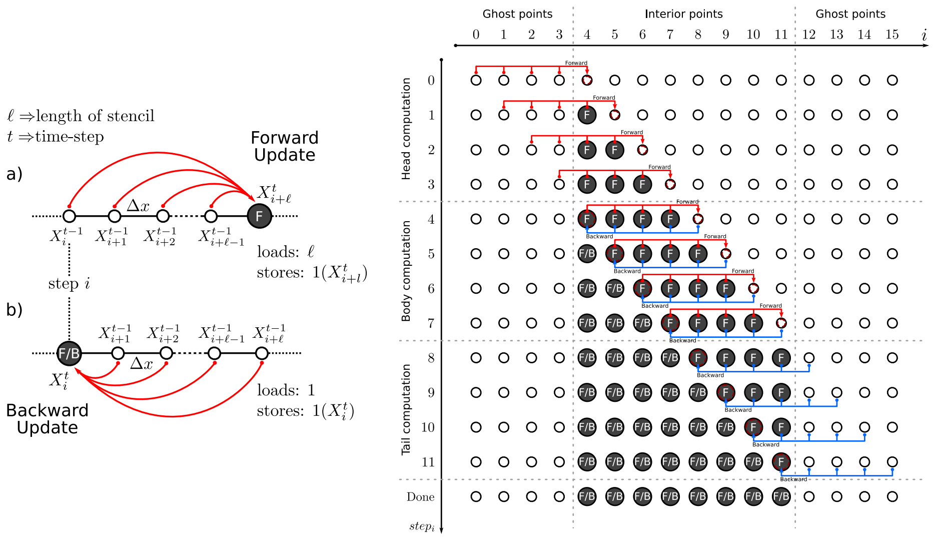

**Fig. 2.** Left: Detail of the two phases for the semi-stencil algorithm at step $i$. a) Forward update on $X^t_{i+\ell}$ point and b) Backward update on $X^t_i$ point. Notice that the $X^{t-1}_{i+1}$ to $X^{t-1}_{i+\ell-1}$ loads can be reused for both phases. Right: Execution example of semi-stencil algorithm in a 1D problem, where $\ell = 4$. $F$ stands for a forward update and $B$ stands for a backward update.

point $X^t_{i+\ell}$ is updated (producing $X'^t_{i+\ell}$) with the $t-1$ rear contributions (Figure 2.Left.a and Equation 3 depicts this operation). In the following equations, prime character denotes the point that has been partially computed.

$$X'^t_{i+\ell} = C_1 * X^{t-1}_{i+\ell-1} + C_2 * X^{t-1}_{i+\ell-2} + \cdots + C_{\ell-1} * X^{t-1}_{i+1} + C_\ell * X^{t-1}_i \tag{3}$$

$$\begin{aligned} X^t_i = X'^t_i &+ C_0 * X^{t-1}_i + C_1 * X^{t-1}_{i+1} + C_2 * X^{t-1}_{i+2} \\ &+ \quad \cdots \quad + C_{\ell-1} * X^{t-1}_{i+\ell-1} + C_\ell * X^{t-1}_{i+\ell} \end{aligned} \tag{4}$$

In the backward update, the $X't_i$ point, computed in a previous forward step, is completed with the front contributions of the axis (points $X^{t-1}_{i+1}$ to $X^{t-1}_{i+\ell}$) (Figure 2.Left.b and Equation 4). Remark, only 1 load is required since almost all data points of time-step $t-1$ were loaded in the forward update. Therefore, this phase needs 1 load for $X^{t-1}_{i+\ell}$ point, 1 store for $X^t_i$ value and finally $\ell + 1$ Multiply-Add instructions ($\ell$ neighbors + self-contribution).

## 4.2 Head, Body and Tail Computations

The FD methods require interior points (inside the solution domain) and ghost points (outside the solution domain). To obtain the correct results on border interior points, the algorithm must be split into three different parts: *head*, *body* and *tail*. The head segment updates the first $\ell$ interior points with the rear contributions (forward phase). In the body segment, the interior points are updated with neighbor interior elements (forward and backward phases). Finally, in the tail segment, the last $\ell$ interior points of the axis are updated with the front contributions (backward phase). Figure 2.Right shows an example execution of the algorithm with the three segments.

## 5 Performance Evaluation and Analysis

We evaluate the performance of the semi-stencil algorithm in two steps: as single SPE implementation, and then integrating the implementation into a numerical kernel of a real-life scientific application. The purpose of the first implementation is to show the local behavior of the algorithm, the second implementation purpose is twofold: measure the expected positive impact of the algorithm on a real application, and expose the impact of the architecture on the algorithm performance. After the evaluation, we analyze the results taking the formulas from Sections 2 and 4 into account.

As we have seen in Section 3, it exists other techniques to deal with stencil computations. Time blocking methods, due to their inherent time dependency, may pose integration problems (boundary conditions, communication, I/O) with scientific numerical kernels. Also, space blocking is not useful on SPM architecture like Cell/B.E., this due to the constant access cost to memory.

Before to present the results, we briefly review the Cell/B.E. architecture. Each Cell/B.E. processor on a QS22 blade contains a general-purpose 64-bit PowerPC-type PPE (3.2 GHz) with 8 GB of RAM. The SPEs have a 128-bit wide SIMD instruction set, which allows to process simultaneously 4 single-precision floating-point operands. Also, the SPEs have software-based SPM called Local Stores (LS) of 256 KB. The 8 SPEs are connected with the PPE by the Element Interconnect Bus of 200 GB/s of peak bandwidth, but the bandwidth to main memory is only 25.6 GB/s.

### Single-SPE Implementation

In the first porting of our semi-stencil algorithm to Cell/B.E. we focus only on the SPE's code, the PPE's code and the techniques required to make the main memory transfers efficient are taking into account in our second semi-stencil implementation. Our implementation is fully handcoded SIMDized (as well as the classical stencil code). The classical stencil implementation takes advantage of optimization techniques such as: software prefetching, software pipelining and loop unrolling.

The following results are expressed in terms of elapsed time and floating-point operations. Notice that the results were computed in single precision arithmetic, and the stencil size ($\ell$) is 4. Our purpose with the results is to compare both approaches implementation. Also, the experiments look for situate the proposed algorithm with respect the peak performance of the Cell/B.E. architecture.

**Table 1.** Performance results for the single SPE implementations. These results were obtained with *IBM XL C/C++ for Multi-core Acceleration for Linux, V9.0* and GCC 4.1.1. No auto-vectorization flags were used, all codes were vectorized by hand. Stencil size $\ell = 4$.

| | Classical Stencil | | Semi-stencil | |
|---|---|---|---|---|
| Compiler (Optimization) | Time [ms] | Performance [GFlops] | Time [ms] | Performance [GFlops] |
| XLC -O3 | 6.84 | 4.32 | 3.35 | 8.83 |
| XLC -O5 | 3.44 | 8.61 | 2.38 | **12.44** |
| GCC -O3 | 7.83 | 3.78 | 4.57 | 6.47 |

The peak performance achieved by the semi-stencil is 12.44 GFlops (Table 1) which corresponds to 49% of the SPE peak performance. Under the same experimental setup the classical stencil reaches 8.61 GFlops (34% of the SPE peak performance). This means that the semi-stencil algorithm is 44% faster than the classical stencil. The projected aggregated performance of this algorithm is 99.52 GFlops for one Cell/B.E., which is to the best of our knowledge [11], the fastest stencil computation on this architecture.

**Table 2.** Pipeline statistics for the single SPE implementations. Obtained with the IBM Full-System Simulator.

| | Classical Stencil | Semi-stencil | Semi-stencil gain |
|---|---|---|---|
| Total cycle count [cycles] | 11460147 | 7592643 | 33.7% |
| CPI [cycles/instructions] | 0.69 | 0.75 | -9% |
| Load/Store instr. [instr.] | 4236607 | 2618079 | 38.2% |
| Floating point instr. | 5800000 | 4652400 | 19.8% |
| Ratio FP/Mem | 1.36 | 1.78 | 24.4% |

In Table 2 every metric is in semi-stencil favor, except the CPI measure, which is 9% better for the classical stencil algorithm. In any case, the most important metric and the one that summarize the performance is the FP/Mem ratio. This ratio is 24% better for the semi-stencil than the classical stencil computation.

## Real-life Scientific Application Implementation

We integrate the semi-stencil implementation with a Reverse-Time Migration (RTM) implementation [5]. RTM is the tool of choice when complex subsurface areas are intended to be clearly defined. RTM has proven to be very useful for the subsaly oil discoveries of the Gulf of Mexico. The core of the RTM is an acoustic wave propagation (PDE+FD) solver, which suits as test case for our proposed algorithm.

The RTM+semi-stencil is 16% faster than RTM+classical, which partially (44% in the ideal case, see Table 1) keeps the performance distance presented

**Table 3.** Performance results for the RTM implementations. These results were obtained with GCC 4.1.1, using only one Cell/B.E. of a QS22 blade. Each experiment carried 500 steps for a 512x512x512 data set. Total time experiments cover both computation and communication times. Stencil size $\ell = 4$.

| | Total | | Only Computation | | Only Communication | Unbalance |
|---|---|---|---|---|---|---|
| Algorithm | Time [s] | Performance [GFlops] | Time [s] | Performance [GFlops] | Time [s] | % |
| RTM+classical | 74.24 | 39.15 | 71.92 | 41.06 | 70.33 | 2.3 |
| RTM+semi-stencil | 63.33 | **46.62** | 48.13 | 61.35 | 64.62 | **25.6** |

in the previous tests. This is because RTM and the data transfers introduce extra complexity to the implementation, RTM has several segments that demand computational resources, e.g. boundary conditions, I/O, etc. As can be seen in Table 3, the unbalance between communication (data transfers from/to main memory to/from LS) thwart the performance. This is especially hard with the RTM+semi-stencil, this implementation lost performance up to 25.6%, even after using multi-buffering technique. If we add this 25.6% to the already gained 16%, we recover the 44% of advantage of the semi-stencil over the classical stencil algorithm.

## Analysis

In this section, given that the experimental results show a clear lead for the semi-stencil algorithm, we confront these results with the theoretical estimation based on the FP/Mem ratio formulas of Section 2 and 4.

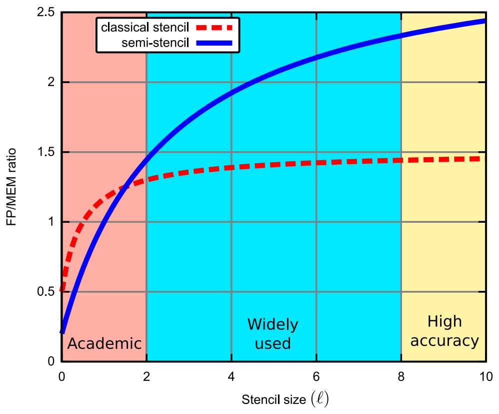

**Fig. 3.** Projected FP/Mem ratio for different stencil size. Notice that the stencils size in the central segment are the most commonly used.

From Figure 3, the theoretical FP/Mem ratio for the classical stencil algorithm is 1.39, while in Table 2 is 1.36. In the semi-stencil case, the theoretical FP/Mem ratio is 1.92 and in Table 2 is 1.78. These figures show that our model of FP/Mem is robust and reliable. Also, both implementation have room for improvements, where for the classical stencil it may only represents a 2.1%. Implementation of the semi-stencil may improve up to 7.3%, but due to its logic complexity this is hard to achieve.

## 6 Conclusions and Future Work

In this paper we have presented a new generic strategy for finite difference stencil computations. This new algorithm approach, called semi-stencil, is especially well suited for scientific applications on heterogeneous architectures, like Cell/B.E., where a low latency SPM exists. Semi-stencil has shown two benefits compared to the classical stencil algorithm:

– It reduces the minimum data working set, reducing the required space in the SPM for each processing unit. This effect becomes more critical for high order stencils.
– It increases data locality, by reducing the number of loads but increasing the number of stores. In a heterogeneous system, these additional stores are executed in the processing unit's SPM, removing the negative impact on the global performance.

For a PDE+FD scheme, the best implementations of the classical stencil computation typically are able to reach to 30% of the processor peak performance. Under the same conditions, the semi-stencil algorithm achieve up to 49% of the peak performance. Also, this improvement revamps the performance of already developed code, as our RTM application.

Future work will focus on research of this novel algorithm on cache-based architectures, where the write policy for the cache hierarchy may have a negative impact on the performance due to the increased number of stores.

## References

1. Brandenburg, A.: Computational aspects of astrophysical MHD and turbulence, vol. 9. CRC, Boca Raton (April 2003)
2. Operto, S., Virieux, J., Amestoy, P., Giraud, L., L'Excellent, J.Y.: 3D frequency-domain finite-difference modeling of acoustic wave propagation using a massively parallel direct solver: a feasibility study. In: SEG Technical Program Expanded Abstracts, pp. 2265–2269 (2006)
3. Alonso, J.L., Andrade, X., Echenique, P., Falceto, F., Prada-Gracia, D., Rubio, A.: Efficient formalism for large-scale ab initio molecular dynamics based on time-dependent density functional theory. Physical Review Letters 101 (August 2008)
4. Groot-Hedlin, C.D.: A finite difference solution to the helmholtz equation in a radially symmetric waveguide: Application to near-source scattering in ocean acoustics. Journal of Computational Acoustics 16, 447–464 (2008)

5. Araya-Polo, M., Rubio, F., Hanzich, M., de la Cruz, R., Cela, J.M., Scarpazza, D.P.: 3D seismic imaging through reverse-time migration on homogeneous and heterogeneous multi-core processors. Scientific Programming: Special Issue on the Cell Processor 16 (December 2008)
6. Kamil, S., Husbands, P., Oliker, L., Shalf, J., Yelick, K.: Impact of modern memory subsystems on cache optimizations for stencil computations. In: MSP 2005: Proceedings of the 2005 Workshop on Memory System Performance, pp. 36–43. ACM Press, New York (2005)
7. Kamil, S., Datta, K., Williams, S., Oliker, L., Shalf, J., Yelick, K.: Implicit and explicit optimizations for stencil computations. In: MSPC 2006: Proceedings of the 2006 Workshop on Memory System Performance and Correctness, pp. 51–60. ACM, New York (2006)
8. Frigo, M., Strumpen, V.: Cache oblivious stencil computations. In: 19th ACM International Conference on Supercomputing, pp. 361–366 (June 2005)
9. Rivera, G., Tseng, C.W.: Tiling optimizations for 3D scientific computations. In: Proc. ACM/IEEE Supercomputing Conference (SC 2000), p. 32 (November 2000)
10. Wonnacott, D.: Time skewing for parallel computers. In: Carter, L., Ferrante, J. (eds.) LCPC 1999. LNCS, vol. 1863, pp. 477–480. Springer, Heidelberg (2000)
11. Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J., Yelick, K.: Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In: SC 2008: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, Piscataway, NJ, USA, pp. 1–12. IEEE Press, Los Alamitos (2008)
12. Allan, V.H., Jones, R.B., Lee, R.M., Allan, S.J.: Software pipelining. ACM Comput. Surv. 27(3), 367–432 (1995)
13. Callahan, D., Kennedy, K., Porterfield, A.: Software prefetching. In: ASPLOS-IV: Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 40–52. ACM, New York (1991)