A Multilevel Parallelization Framework for High-Order Stencil Computations

Hikmet Dursun, Ken-ichi Nomura, Liu Peng, Richard Seymour,
Weiqiang Wang, Rajiv K. Kalia, Aiichiro Nakano, and Priya Vashishta

Collaboratory for Advanced Computing and Simulations,
Department of Computer Science,
Department of Physics & Astronomy,
Department of Chemical Engineering & Materials Science,
University of Southern California, Los Angeles, CA 90089-0242, USA
{hdursun, knomura, liupeng, rseymour, wangweiq, rkalia, anakano, priyav}@usc.edu

Abstract. Stencil based computation on structured grids is a common kernel to broad scientific applications. The order of stencils increases with the required precision, and it is a challenge to optimize such high-order stencils on multicore architectures. Here, we propose a multilevel parallelization framework that combines: (1) inter-node parallelism by spatial decomposition; (2) intra-chip parallelism through multithreading; and (3) data-level parallelism via single-instruction multiple-data (SIMD) techniques. The framework is applied to a 6^{th} order stencil based seismic wave propagation code on a suite of multicore architectures. Strong-scaling scalability tests exhibit superlinear speedup due to increasing cache capacity on Intel Harpertown and AMD Barcelona based clusters, whereas weak-scaling parallel efficiency is 0.92 on 65,536 BlueGene/P processors. Multithreading+SIMD optimizations achieve 7.85-fold speedup on a dual quad-core Intel Clovertown, and the data-level parallel efficiency is found to depend on the stencil order.

Keywords: Stencil computation, multithreading, single instruction multiple data parallelism, message passing, spatial decomposition.

1 Introduction

Design complexity and cooling difficulties in high-speed single-core chips have forced chip makers to adopt a multicore strategy in designing heterogeneous hardware architectures [1-3]. The shift in architectural design has provided incentives for the high-performance computing (HPC) community to develop a variety of programming paradigms that maximally utilize underlying hardware for broad computational applications [4].

A common core computational kernel used in a variety of scientific and engineering applications is stencil computation (SC). Extensive efforts have been made to optimize SC on multicore platforms with the main focus on low-order SC. For example, Williams et al. [5] have optimized a lattice Boltzmann application on

leading multicore platforms, including Intel Itanium2, Sun Niagara2 and STI Cell. Datta et al. have recently performed comprehensive SC optimization and auto-tuning with both cache-aware and cache-oblivious approaches on a variety of state-of-the-art architectures, including NVIDIA GTX280, etc [6]. Other approaches to SC optimization include methods such as tiling [7] and iteration skewing (if the iteration structure allows it) [8-10]. Due to the importance of high-order SC in the broad applications and the wide landscape of multicore architectures as mentioned above, it is desirable to develop a unified parallelization framework and perform systematic performance optimization for the high-order SC on various multicore architectures.

In this paper, we propose a multilevel parallelization framework addressing high-order stencil computations. Our framework combines: (1) inter-node parallelism by spatial decomposition; (2) intra-chip parallelism through multithreading; and (3) data-level parallelism via single-instruction multiple-data (SIMD) techniques. We test our generic approach with a 6^{th} order stencil based seismic wave propagation code on a suite of multicore architectures. Strong-scaling scalability tests exhibit superlinear speedup due to increasing cache capacity on Intel Harpertown and AMD Barcelona based clusters, whereas weak-scaling parallel efficiency is 0.92 on 65,536 BlueGene/P processors. Our intra-core optimizations combine Multithreading+SIMDization approaches to achieve 7.85-fold speedup on a dual quad-core Intel Clovertown. We also explore extended precision high order stencil computations at 7^{th} and 8^{th} orders to show dependency of data-level parallel efficiency on the stencil order.

This paper is organized as follows: An overview of the stencil problem is given in Section 2 together with the description of experimental application. Section 3 presents the parallelization framework and details its levels. Section 4 describes the suite of experimental platforms, with details on the input decks used and the methodology used to undertake inter-node and intra-node scalability analysis. Conclusions from this work are discussed in Section 5.

2 High-Order Stencil Application

This section introduces the general concept of high-order stencil based computation as well as details of our experimental application, i.e., seismic wave propagation code.

2.1 Stencil Computation

Stencil computation (SC) is at the heart of a wide range of scientific and engineering applications. A number of benchmark suites, such as PARKBENCH [11], NAS Parallel Benchmarks [12], SPEC [13] and HPFBench [14] include stencil computations to evaluate performance characteristics of HPC clusters. Implementation of special purpose stencil compilers highlights the common use of stencil computation based methods [15]. Other examples to applications employing SC include thermodynamically and mechanically driven flow simulations, e.g. oceanic circulation modeling [16], neighbor pixel based computations, e.g.

multimedia/image-processing [17], quantum dynamics [18] and computational electromagnetics [19] software using the finite-difference time-domain method, Jacobi and multigrid solvers [10].

SC involves a field that assigns values $v_t(\mathbf{r})$ to a set of discrete grid points $\mathbf{\Omega} = \{\mathbf{r}\}$, for the set of simulation time steps $T = \{t\}$. SC routine sweeps over $\mathbf{\Omega}$ iteratively to update $v_t(\mathbf{r})$ using a numerical approximation technique as a function of the values of the neighboring nodal set including the node of interest, $\mathbf{\Omega}' = \{\mathbf{r}' \mid \mathbf{r}' \in neighbor(\mathbf{r})\}$ which is determined by the stencil geometry. The pseudocode below shows a naïve stencil computation:

for $\forall t \in T$
  for $\forall \mathbf{r} \in \mathbf{\Omega}$
    $v_{t+1}(\mathbf{r}) \leftarrow f(\{v_t(\mathbf{r}') \mid \mathbf{r}' \in neighbor(\mathbf{r})\})$

where f is the mapping function and v_t is the scalar field at time step t. SC may be classified according to the geometric arrangement of the nodal group $neighbor(\mathbf{r})$ as follows: First, the order of a stencil is defined as the distance between the grid point of interest, $\mathbf{r}$, and the farthest grid point in $neighbor(\mathbf{r})$ along a certain axis. (In a finite-difference application, the order increases with required level of precision.) Second, we define the size of a stencil as the cardinality $|\{\mathbf{r}' \mid \mathbf{r}' \in neighbor(\mathbf{r})\}|$, i.e., the number of grid points involved in each stencil iteration. Third, we define the footprint of a stencil by the cardinality of minimum bounding orthorhombic volume, which includes all involved grid points per stencil. For example, Fig. 1 shows a 6th order, 25-point SC whose footprint is $13^2 = 169$ on a 2-dimensional lattice. Such stencil is widely used in high-order finite-difference calculations [20, 21].

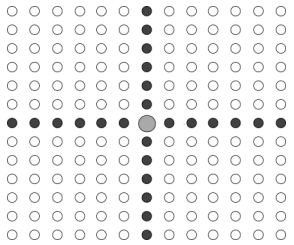

Fig. 1. 6-th order, 25-point SC whose footprint is 13^2 on a 2-dimensional lattice. The grid point of interest, $\mathbf{r}$, is shown as the central grey circle while the set of neighbor points, excluding $\mathbf{r}$, i.e., $\{\mathbf{r}' \mid \mathbf{r}' \in neighbor(\mathbf{r})\} - \{\mathbf{r}\}$, is illustrated as solid circles. White circles show the other lattice sites within the stencil footprint, which are not used for calculation of $v(\mathbf{r})$.

2.2 A High-Order Stencil Application for Seismic Wave Propagation

This subsection introduces our experimental application that simulates seismic wave propagation by employing a 3D equivalent of the stencil in Fig. 1 to compute spatial derivatives on uniform grids using a finite difference method. The 3D stencil kernel is highly off-diagonal (6-th order) and involves 37 points (footprint is $13^3 = 2{,}197$), i.e., each grid point interacts with 12 other grid points in each of the x, y and z Cartesian directions.

A typical problem space of the program includes several hundreds of grid nodes in each dimension amounting to an aggregate amount of millions of grid points. Our largest experiment has 400^3 = 64 million points. For each grid point, the code allocates 5 floats to hold temporary arrays and intermediate physical quantities and the result, therefore its memory footprint is $5\times4\text{bytes}\times400^3$ = 1.28GB. Thus, the application not only involves heavy computational requirements but also needs three orders of magnitude more memory in comparison with the cache size offered by multicore architectures in the current HPC literature.

3 Multilevel Parallelization Framework

This section discusses inter-node and intra-node optimizations we use and outlines our parallelization framework that combines: (1) inter-node parallelism by spatial decomposition; (2) intra-chip parallelism through multithreading; and (3) data-level parallelism via SIMD techniques.

3.1 Spatial Decomposition Based on Message Passing

Our parallel implementation essentially retains the computational methodology of the original sequential code. All of the existing subroutines and the data structures are retained. However, instead of partitioning the computational domain, we divide the 3D data space to a mutually exclusive and collectively exhaustive set of subdomains and distribute the data over the cluster assigning a smaller version of the same problem to each processor in the network, then employ an owner-computes rule.

Subdomains should have the same number of grid points to balance the computational load evenly. However, a typical stencil computation not only uses the grid points owned by the owner processor, but also requires boundary grid points to be exchanged among processors through a message passing framework. Therefore each subdomain is augmented with a surrounding buffer layer used for data transfer.

In a parallel processing environment, it is vital to understand the time complexity of communication and computation dictated by the underlying algorithm, to model performance of parallelism. For a d-dimensional finite difference stencil problem of order m and global grid size n run on p processors, the communication complexity is $O(m(n/p)^{(d-1)/d})$, whereas computation associated by the subdomain is proportional to the number of owned grid points, therefore its complexity is $O(mn/p)$ (which is linear in m, assuming a stencil geometrically similar to that shown in Fig.1). For a 3D problem, they reduce to $O(m(n/p)^{2/3})$ and $O(mn/p)$, which are proportional to the surface area and volume of the underlying subdomain, respectively. Accordingly, subdomains are selected to be the optimal orthorhombic box in the 3D domain minimizing the surface-to-volume ratio, $O((p/n)^{1/3})$, for a given number of processors. This corresponds to a cubic subvolume if processor count is cube of an integer number; otherwise a rectangular prism shaped subset is constructed by given logical network topology. Fig. 2(a) shows the decomposition of problem domain into cubical subvolumes, which are assigned to different processors.

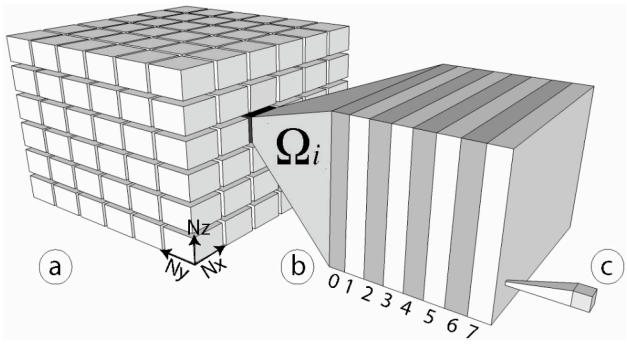

Fig. 2. Schematic of multilevel parallelization. In the uppermost level of multilevel parallelization, the total simulation space $\mathbf{\Omega}$ is decomposed into several spatial sub-domains $\mathbf{\Omega}_i$ of size $N_x \times N_y \times N_z$ points in corresponding directions where $\mathbf{\Omega} = \cup \mathbf{\Omega}_i$ and each domain is mapped onto a node in the cluster (a). Intra-node parallelization includes POSIX threading to exploit all available cores in the underlying multicore processor (b). It shows 8 threads and numerals as their IDs, which is intended for dual quad-core architecture nodes. Data-level parallelism constitutes the last level in the hierarchical parallelization methodology which we achieve through vectorization and implement by Streaming SIMD Extensions (SSE3) at Intel based architectures (c).

3.2 Intra-node Parallelism Based on Multithreading

On a cluster of Intel Clovertown, the code is implemented based on hierarchical spatial decomposition: (1) inter-node parallelization with the upper-level spatial decomposition into domains based on message passing; and (2) intra-node parallelization with the lower-level spatial decomposition within each domain through multithreading.

Fig. 3 shows the pseudocode for the threaded code section. There are 3 subroutines implementing this pseudocode in the actual application to perform computations in each of 3D Cartesian directions (represented by `toCompute` in the pseudocode) at each simulation step. 1D data array (`current` in Fig. 3) for grid points allocates x data to be unit stride direction, and y data to be N_x stride, finally z data with $N_x \times N_y$ stride. Threads spawned in `toCompute` direction evenly partition the domain in `lowStride` dimension and apply stencil formula for the partial spatial derivation with respect to the corresponding Cartesian coordinate, i.e., `toCompute`. Therefore, for a thread spawned to perform calculations in the z direction, `toCompute` is the z direction whereas `lowStride` corresponds to the x direction since y stride is larger than the unit stride for x. Before the actual computation is performed, consecutive points in `toCompute` direction, which are stride away in the `current` array, are packed into a temporary array, named `tempCurrent` in Fig. 3, to reduce effective access time to the same data throughout the computation. (This only applies to the implementations in non-unit access directions, y and z, as x data are already consecutive in `current`.) Finally, `tempCurrent` is traversed to compute new field values to update `next` array. Threads of the same direction can independently execute and perform updates on their assigned subdomain without

introducing any locks until finally they are joined before the stencil for the other directions are calculated.

```
FOR each gridPoint in Ωi in highStride direction
   FOR each gridPoint in Ωi in lowStride direction
      SET packedGrid to 0
      FOR each gridPoint in Ωi in toCompute direction
          STORE current[gridPoint] in
          tempCurrent[packedGrid]
          INCREMENT packedGrid
      END FOR
      FOR each packedGridPoint in tempCurrent
          COMPUTE next[gridPoint] as the accumulation of
                  packedGridPoint contributions
      END FOR
   END FOR
END FOR
```

Fig. 3. Pseudocode for the threaded code section. Since threads store their data in the `next` array, they avoid possible race condition and the program can exploit thread-level parallelism (TLP) in Intel Clovertown architecture.

3.3 Single Instruction Multiple Data Parallelism

Most modern computing platforms have incorporated single-instruction multiple-data (SIMD) extensions into their processors to exploit the natural parallelism of applications if the data can be SIMDized (i.e., if a vector instruction can simultaneously operate on a sequence of data items.). On Intel quadcore architectures, 128-bit wide registers can hold four single precision (SP) floating point (FP) numbers that are operated concurrently, and thus throughput is 4 SP FP operations per cycle and the ideal speedup is 4. However, determining how instructions depend on each other is critical to determine how much parallelism exists in a program and how that parallelism can be exploited. In particular, for finite-difference computations, instructions operating on different grid points are independent and can be executed simultaneously in a pipeline without causing any stalls. Because of this property, we can expose a finer grained parallelism through vectorization on top of our shared-memory Pthreads programming and manually implement using Streaming SIMD Extensions (SSE3) intrinsics on Intel based architectures, whose prototypes are given in Intel's xmmintrin.h and pmmintrin.h. In our SIMDization scheme, we first load successive neighbor grid points and corresponding stencil coefficients to registers using _mm_load_ps. Secondly, we perform computations using arithmetic SSE3 intrinsics (e.g. _mm_add_ps, _mm_mul_ps) and reduce the results by horizontally adding the adjacent vector elements (_mm_hadd_ps). Finally, we store the result back to the data array (_mm_store_ss). Our results have exhibited sub-optimal performance gain for the stencil orders not divisible by 4, as unused words in 4 word registers were padded with zeros, albeit SIMDization indeed contributed in intra-node speedup combined with multithreading for such stencil orders as well. More detailed performance-measurement results are outlined in Sections 4.3 and 4.4.

4 Performance Tests

The scalability and parallel efficiency of the seismic stencil code have been tested on various high-end computing platforms including dual-socket Intel Harpertown and Clovertown, AMD Barcelona based clusters at the high performance computing communications facility of University of Southern California (HPCC-USC). We also present weak-scaling benchmark results at 65,536 IBM BlueGene/P processors at the Argonne National Laboratory. The following subsections provide the details of our benchmarks and present the results.

4.1 Weak-Scaling

We have implemented the spatial decomposition using the message passing interface (MPI) [22] standard and have tested the inter-node scalability on IBM BlueGene/P and an HPCC-USC cluster. Fig. 4(a) shows the running and communication times of the seismic code at HPCC-USC, where each node has Intel dual quad-core Xeon E5345 (Clovertown) processors clocked at 2.33 GHz, featuring 4 MB L2 cache per die, together with 12 GB memory and are connected via 10-gigabit Myrinet. Here, we scale the number of grid points linearly with the number of processors: $10^6 p$ grid points on p processors. The wall-clock time per time step approximately stays constant as p increases. The weak-scaling parallel efficiency, the running time on 1 processor divided by that on p processors, is observed to be 0.992 on the Clovertown based cluster for our largest run on 256 processors. Fig. 4(b) also shows good weak-scaling parallel efficiency with increased grain size (8×10^6) on 65,536 BlueGene/P processors, where each node has 2 GB DDR2 DRAM and four 450 POWER PC processors clocked at 850 MHz, featuring a 32 KB instruction and data cache, a 2 KB L2 cache and a shared 8 MB L3 cache. Spatial decomposition is thus highly effective in terms of inter-node scalability of stencil computations up to 65,536 processors.

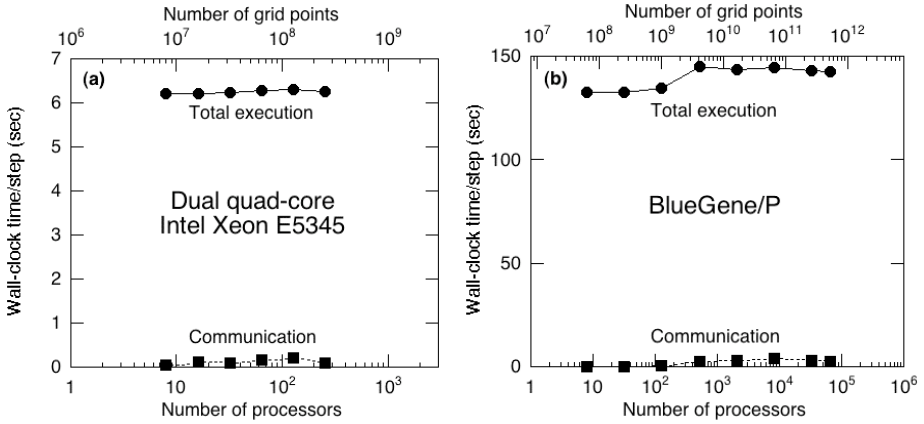

Fig. 4. Weak-scaling performance of spatial decomposition methodology, where the problem size scales: (a) smaller granularity (20 MB/process) on a dual quadcore Intel Xeon E5345 cluster; (b) larger granularity (160 MB/process) on BlueGene/P.

4.2 Strong-Scaling

Strong-scalability (i.e., achieving large speedup for a fixed problem size on a large number of processors) is desirable for many finite-difference applications, especially for simulating long-time behaviors. Here, strong-scaling speedup is defined as the ratio of the time to solve a problem on one processor to that on p processors. Fig. 5 shows strong-scaling speedups for a fixed global problem size of 400^3 on several AMD and Intel quadcore based clusters. We observe superlinear speedups for increasing processor count on both AMD and Intel architectures, which may be interpreted as a consequence of the increasing aggregate cache size as explained below.

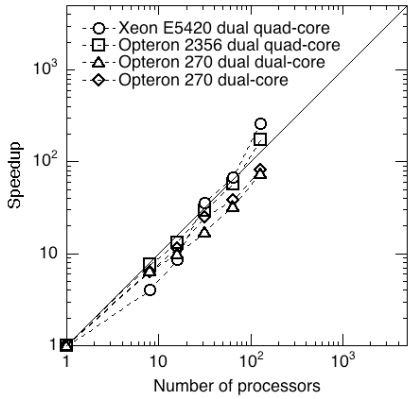

Fig. 5. Strong-scaling benchmark on various multicore CPUs. The speedup over single-processor run time is plotted as a function of the number of processors p. Intel Xeon E5420 shows superlinear speedup for smaller p as the aggregate cache size increases faster because of its 12 MB L2 cache. The solid line shows ideal speedup.

The original seismic stencil code suffers from high effective memory access time when the entire problem space is put into memory of one processor. We have analyzed cache and TLB misses for the original code by using Intel's Vtune Performance Analyzer and found that, for high-stride access computation that implements the partial spatial derivative in the z direction, the duration of page-walks amounts to more than 25% of core cycles throughout the thread execution, implying a high TLB miss rate resulting in greater effective memory access time. As we divide the problem space, the effective access time reduces since the hit ratio is higher for smaller problem size, even though the latency to the cache remains the same. It should be noted that, in our strong-scaling experiment, not only the processor count is increasing but also the size of aggregate caches from processors. With larger aggregate cache size, more (or even entire, depending on processor count) data can fit into the caches. The cache effect is most pronounced on Intel's Xeon E5420 (Harpertown) in Fig. 5. Intel's second-generation quadcore processor Harpertown features a shared 6 MB L2 cache per chip that accommodates two cores. This amounts to 12 MB of cache per multi-chip module (MCM), 6 times more than 2 MB (4×512KB) L2 cache offered by AMD Opteron 2356 (Barcelona). As a result,

Harpertown crosses over to the superlinear-scaling regime on a smaller number of processors compared with that for Barcelona (see Fig. 5).

4.3 Intra-node Optimization Results

In addition to the massive inter-node scalability demonstrated above, our framework involves the lower levels of optimization: First, we use multithreading explained in Section 3.2, implemented with Pthreads. Next, we vectorize both `STORE` and `COMPUTE` statements inside the triply nested for loops in Fig. 3 by using SSE3 intrinsics on the dual Intel Clovertown platform. We use Intel C compiler (icc) version 9.1 for our benchmarks.

Fig. 6(a) shows the reduction in clock time spent per simulation step due to multithreading and SIMDization optimizations. Corresponding speedups are shown in Fig. 6(b). To delineate the performances of multithreading and SIMDization approaches, we define a performance metric as follows. We use the clock time for one simulation step of single threaded, non-vectorized code to be the sequential run time T_s. We denote the parallel run time, $T_p(NUM\_THREADS)$, to be the clock time required for execution of one time step of the algorithm as a function of spawned thread number, in presence of both multithreading and SIMD optimizations. Then combined speedup, S_c, shown in Fig. 6(b) is equal to the ratio T_s/T_p as a function of thread number. We remove SIMD optimizations to quantify the effect of multithreading only, and measure the parallel running times for a variety of thread numbers, and state the multithreading speedup, S_t, with respect to T_s. Finally, we attribute the excess speedup, S_c/S_t, to SIMD optimizations.

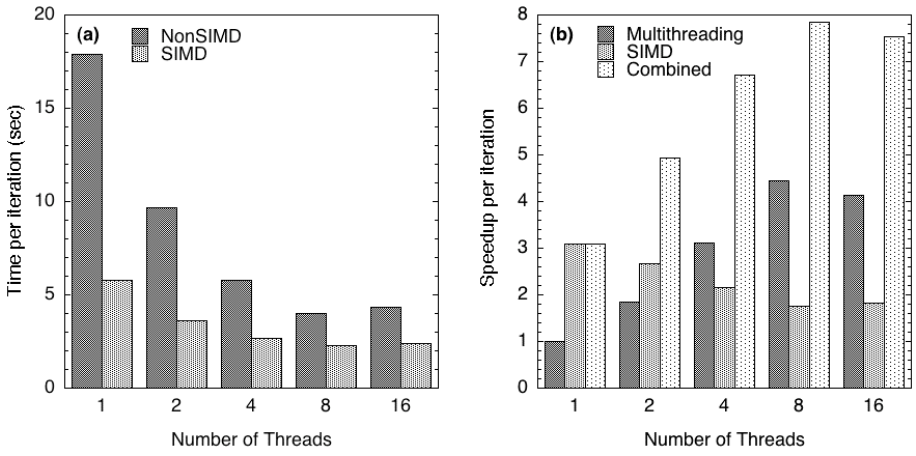

Fig. 6. The wall-clock time per iteration for non SIMD and SIMDized codes (a) and breakdown of speedups (due to multithreading and data-level parallelism along with the combined intra-node speedup) (b) as a function of the number of threads on dual quad-core Intel Clovertown. The best observed intra-node speedup is 7.85 with 8 threads spawned on 8 cores.

Fig. 6(b) shows the best speedup of 7.85 for 8 threads on a dual quadcore Intel Clovertown node. This amounts to 7.85/8 = 0.98 intra-node parallel efficiency, even though separate contributions of SIMDization and multithreading are less than ideal. Multithreading speedup increases until thread number is equal to the available cores, after which we see performance degradation in the SIMD speedup due to more frequent exchanges of data between vector registers and main memory in case of greater number of threads.

4.4 Effects of Stencil Order

We have also studied the effect of stencil order on intra-node parallel efficiency. As mentioned in Section 3.3, our SIMDization scheme is suboptimal for our seismic stencil code, for which the stencil order is not a multiple of 4. In fact, it is 6 and uses two quad-word registers by padding one of them with 2 zeros. To reduce this overhead, we have increased the precision of our simulation and performed a similar computation with 7^{th} order stencil (1 zero padding at one of the registers) and 8^{th} order stencil (utilizing all 4 words of both of the 128-bit registers).

Fig. 7(a) shows the dependency of the combined multithreading+SIMD speedup, as defined in previous subsection, on stencil sizes. When all SIMD vectors are saturated, i.e., fully used, the best combined speedup is observed. In our implementation, SIMD vectors are saturated for 8^{th} order stencil, and best speedup is observed with 8 threads on dual quadcore Intel Clovertown. In fact, we have achieved superlinear intra-node speedup of 8.7 with 8 threads on 8 cores. (Since we use in-core optimizations as well, it is not surprising to see more than 8-fold speedup.)

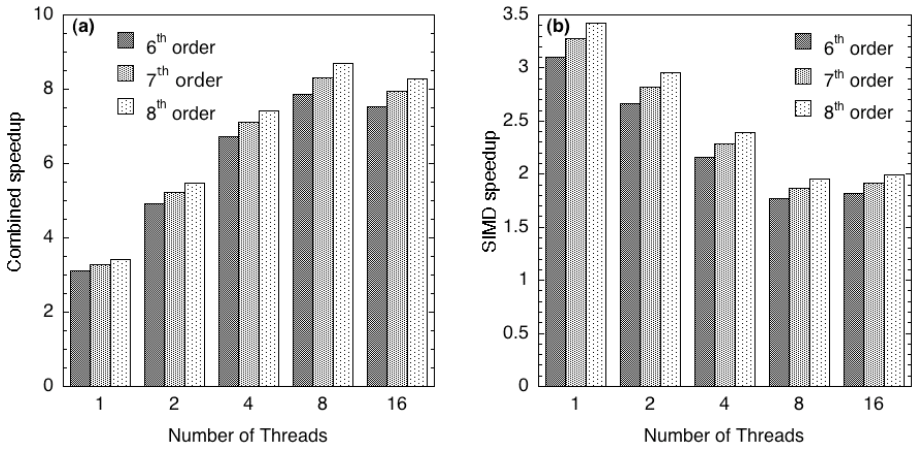

Fig. 7. Speedups for higher order stencils, which are required for higher precision in finite-differences calculation. Combined multithreading+SIMD (a) and SIMD (b) speedups are shown as a function of the number of threads for 6^{th}, 7^{th} and 8^{th} order stencils.

Fig. 7(b) shows the data parallelism contribution in the combined speedup. It normalizes the combined speedup in Fig. 7(a) with multithreaded, but non-vectorized

speedup in order to quantify the speedup due to data level parallelism only. Saturating vectors with actual data is observed to yield the highest performance gain, which corresponds to 8^{th} order stencil in our implementation. We have observed 3.42-fold speedup due to data-level parallelism in 1 thread run for 8^{th} order stencil, pointing out the highest SIMD efficiency of 3.42/4 = 0.85 in Fig. 7(b). The figure also confirms the decreasing performance of SIMD as a function of thread count, as shown in Fig. 6(b).

5 Conclusions

In summary, we have developed a multilevel parallelization framework for high order stencil computations. We have applied our approach to a 6^{th}-order stencil based seismic wave propagation code on a suite of multicore architectures. We have achieved superlinear strong-scaling speedup on Intel Harpertown and AMD Barcelona based clusters through the uppermost level of our hierarchical approach, i.e., spatial decomposition based on message passing. Our implementation has also attained 0.92 weak-scaling parallel efficiency at 65,536 BlueGene/P processors. Our intra-node optimizations included multithreading and data-level parallelism via SIMD techniques. Multithreading+SIMD optimizations achieved 7.85-fold speedup and 0.98 intra-node parallel efficiency on dual quadcore Intel Clovertown platform. We have quantified the dependency of data-level parallel efficiency on the stencil order, and have achieved 8.7-fold speedup for an extended problem employing 8^{th} order stencil. Future work will address the analysis and improvement of flops performance, and perform lower-level optimizations at a broader set of emerging architectures.

Acknowledgments. This work was partially supported by Chevron—CiSoft, NSF, DOE, ARO, and DTRA. Scalability and performance tests were carried out using High Performance Computing and Communications cluster of the University of Southern California and the BlueGene/P at the Argonne National Laboratory. We thank the staff of the Argonne Leadership Computing Facility for their help on the BlueGene/P benchmark.

References

1. Dongarra, J., Gannon, D., Fox, G., Kennedy, K.: The impact of multicore on computational science software. CTWatch Quarterly **3** (2007) 11-17
2. Barker, K.J., Davis, K., Hoisie, A., Kerbyson, D.J., Lang, M., Pakin, S., Sancho, J.C.: Entering the petaflop era: the architecture and performance of Roadrunner. Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, Austin, Texas (2008)
3. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Pishker, W.L., Shalf, J., Williams, S.W., Yelick, K.A.: The landscape of parallel computing research: a view from Berkeley. University of California, Berkeley (2006)
4. Pakin, S.: Receiver-initiated message passing over RDMA networks. Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium, Miami, Florida (2008)

5. Williams, S., Carter, J., Oliker, L., Shalf, J., Yelick, K.: Lattice Boltzmann simulation optimization on leading multicore platforms. Proceedings of the 22nd International Parallel and Distributed Processing Symposium, Miami, Florida (2008)
6. 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. Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, Austin, Texas (2008)
7. Rivera, G., Tseng, C.-W.: Tiling optimizations for 3D scientific computations. Proceedings of the 2000 ACM/IEEE conference on Supercomputing. IEEE Computer Society, Dallas, Texas (2000)
8. Frigo, M., Strumpen, V.: Cache oblivious stencil computations. Proceedings of the 2005 ACM/IEEE conference on Supercomputing. ACM, Cambridge, Massachusetts (2005)
9. Wonnacott, D.: Using time skewing to eliminate idle time due to memory bandwidth and network limitations. Proceedings of the 14th IEEE International Parallel and Distributed Processing Symposium, Cancun, Mexico (2000)
10. Renganarayanan, L., Harthikote-Matha, M., Dewri, R., Rajopadhye, S.V.: Towards optimal multi-level tiling for stencil computations. Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium, Long Beach, California (2007)
11. PARKBENCH: PARallel Kernels and BENCHmarks. Available from http://www.netlib.org/parkbench
12. Kamil, S., Datta, K., Williams, S., Oliker, L., Shalf, J., Yelick, K.: Implicit and explicit optimizations for stencil computations. Proceedings of the 2006 Workshop on Memory System Performance and Correctness. ACM, San Jose, California (2006)
13. Schreiber, R., Dongarra, J.: Automatic blocking of nested loops. Technical Report, University of Tennessee (1990)
14. Desprez, F., Dongarra, J., Rastello, F., Robert, Y.: Determining the idle time of a tiling: new results. Journal of Information Science and Engineering **14** (1998) 167-190
15. Bromley, M., Heller, S., McNerney, T., Steele, J.G.L.: Fortran at ten gigaflops: the connection machine convolution compiler. Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation. ACM, Toronto, Ontario, Canada (1991)
16. Bleck, R., Rooth, C., Hu, D., Smith, L.T.: Salinity-driven Thermocline Transients in a Wind- and Thermohaline-forced Isopycnic Coordinate Model of the North Atlantic. Journal of Physical Oceanography **22(12)** (1992) 1486–1505
17. Harlick, R., Shapiro, L.: Computer and Robot Vision. Addision Wesley (1993)
18. Nakano, A., Vashishta, P., Kalia, R.K.: Multiresolution molecular dynamics algorithm for realistic materials modeling on parallel computers. Computer Physics Communications **83** (1994)
19. Taflove, A., Hagness, S.C.: Computational Electrodynamics: The Finite-Difference Time-Domain Method. 3rd edn. Artech House, Norwood, Massachusetts (2005)
20. Shimojo, F., Kalia, R.K., Nakano, A., Vashishta, P.: Divide-and-conquer density functional theory on hierarchical real-space grids: Parallel implementation and applications. Physical Review B **77** (2008)
21. Stathopoulos, A., Öğüt, S., Saad, Y., Chelikowsky, J.R., Kim, H.: Parallel methods and tools for predicting material properties. Computing in Science and Engineering **2** (2000) 19-32
22. Snir, M., Otto, S.: MPI-The Complete Reference: The MPI Core. MIT Press (1998)