High Performance CPU Kernels for Multiphase Compressible Flows

Babak Hejazialhosseini, Christian Conti, Diego Rossinelli, and Petros Koumoutsakos

Computational Science and Engineering Lab, ETH Zurich, Zurich, Switzerland
{hbabak,cconti,diegor,petros}@mavt.ethz.ch

Abstract. We develop efficient CPU kernels for multiphase compressible flows and evaluate different optimization strategies. The presented software achieves up to 48% of the peak performance on shared memory architectures, outperforming by 9-14X what is considered to be state-of-the-art. On 48-core CPUs we observe speedups of 40-45X and measure up to 360 GFLOP/s over 840 GFLOP/s of the peak.

1 Introduction

Simulations of multiphase compressible flows are a critical testbed for high performance computing as they face performance issues that are often encountered in other branches of computational science. Such simulations are essential for studies of shock wave lithotripsy and combustion, problems that do not easily render themselves to experimental studies. Numerical investigations have therefore become an established approach in studying such complex flows [1]. Among the most modern techniques, we find adaptive mesh refinement (AMR) [2] and wavelet-based adaptive grids [3], which concentrate the computation on regions of interest. Although these techniques provide substantial algorithmic improvements, their performance impact in terms of GFLOP/s and hardware utilization (i.e. fraction of the peak performance) has not been extensively reported except for a few cases limited to heterogenous multicore/GPUs platforms [4].

In contrast to synthetic benchmarks, for real world applications harnessing the full potential of current state-of-the-art multicores has been shown to be hardly possible as they reach only about 1% of the peak [5]. A primary cause of this issue is the imbalance between the peak performance and the system memory bandwidth (1-10 TFLOP/s versus 0.1-1 TB/s) meaning that applications should employ kernels exhibiting ratios of 10 FLOP/B (bytes of off-chip memory traffic) or more to achieve peak performance. As discussed in this work, such high ratios in the context of multiphase compressible flows are not always realistic and, in the few cases they are, reaching them requires revisiting both algorithms and memory layouts.

Optimization techniques have been proposed to address this widespread challenge ranging from software autotuning tools [6] to automated code generation [7] and hardware/software co-design for domain specific problems [8]. Other successful optimization techniques [9] are based on the roofline model [10].

A number of open issues can be identified in the context of fast simulations for compressible flows. Firstly, there is poor analysis on the performance expectation, which helps contextualizing the measurements. Secondly, it is not clear how the performance of AMR solvers would compare to a highly optimized uniform resolution solver. Thirdly, to the best of our knowledge, for simulations of compressible flows there are no standard packages to assist the software development (such as "High-Performance Linpack" (HPL) [11]) whose performance can reach a significant fraction of the peak: reported per-core performance of the most compute intensive kernel (6000 FLOP/grid point) of the fastest existing software for compressible flows indicate results in the range of 2-3% of the peak per-core performance [12,13].

In this work, we evaluate a set of optimization techniques for the simulation of multiphase compressible flows that is tailored to state-of-the-art multicore platforms and discuss the measured performance. Furthermore, we provide an a-priori analysis on the expected performance. The paper is organized as follows. In Section 2 we describe the governing equations and the considered numerical schemes. In Section 3 we identify the performance bottlenecks in the solver and we discuss the data structures and techniques adopted in the software used to mitigate these barriers. In Section 4 we analyze and discuss the resulting performance for the simulations of the shock-bubble interaction.

2 Governing Equations and Numerical Methods

We model an inviscid multiphase compressible flow described by the Euler equations using the one-fluid formulation [14]:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{u}), \tag{1}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} = -\nabla \cdot (\rho \mathbf{u}\mathbf{u}^T - p\mathbb{I}), \tag{2}$$

$$\frac{\partial E}{\partial t} = -\nabla \cdot ((E + p)\mathbf{u}), \tag{3}$$

$$\frac{\partial \phi}{\partial t} = -\mathbf{u} \cdot \nabla \phi, \tag{4}$$

with ρ being the density, $\mathbf{u}$ the velocity vector, p the pressure, E the total energy of the fluid and ϕ the interface marker function. To close the system of equations, we assume that the fluid follows the ideal gas equation of state,

$$p = (\gamma - 1)(E - \frac{1}{2}\rho|\mathbf{u}|^2), \tag{5}$$

with γ being the ratio of specific heats of each phase.

We use the finite volume discretization and solve the integral form of Equations (1-4) by reconstructing flow quantities and computing the numerical flux on the finite volume cell interfaces. The computation and summation of fluxes are referred to as the computation of the right hand side (RHS) and are coupled to the low-storage third-order TVD Runge-Kutta time stepping scheme.

One simulation step consists of the following stages: estimation of the time step, conversion of conserved to primitive quantities, computation of the RHS and update of the conserved quantities. The RHS computation is itself composed of a series of substeps consisting in 5th order WENO reconstructions [15], evaluations of the HLLE numerical fluxes [16] and summation of the fluxes.

3 Design and Techniques

Previous works indicate that the computation of the RHS is the most expensive part of a simulation step [4]. Furthermore, its *Operational Intensity* (OI), i.e. the ratio of FLOP to bytes of off-chip memory transfers, is low and therefore it implies that in order to observe decent performance, we need to increase data locality. A low OI means that the OI is below the ratio of peak performance to peak memory bandwidth and indicates that the kernel performance is bound by the memory bandwidth of the system that runs it.

Our compressible flow simulations rely on a "software stack" with two software layers, namely core and node layers. This modular structure separates the optimization "domains" as it is directly related to the underlying hardware and increase flexibility and code reusability. These separations in turn facilitate the software development for new applications, techniques and hardware. Optimizations that affect the operational intensity are in general applied to the core layer whereas optimizations related to thread level parallelism and communication are covered in the node layer.

Data Structures. The governing equations considered in this work involve 6 unknowns, ρ, u, v, w, E and ϕ, which are organized into an Array of Structures (AoS) format referred to as *grid point*, to maintain maximum software flexibility. To increase the data locality we introduce an intermediate data structure, called hereafter *block*, which contains 16-32 grid points per dimension. In order to further enforce data locality, Morton space-filling is used to index the blocks. A secondary advantage of introducing blocks is that any technique developed here can be employed in block-based AMR solvers as well. The main downside comes from the extra overhead of replicating some grid points (ghosts) required to process the blocks.

Core Layer. The core layer maps to the processor cores and is thus responsible for Data Level Parallelism (DLP) and Instruction Level Parallelism (ILP).

As we expect the performance of the kernels to be memory-bound, all data structures are properly 16- or 32-byte aligned. Our design is also oriented towards data and computation reordering: temporal locality is improved by processing blocks slice-by-slice and the memory footprint is reduced by computing on ring-like buffers composed of a few slices, analogously to [17].

Table 1. Arithmetic and operational intensities

Kernel	AI [FLOP/B]	OI [FLOP/B]
dt	1.7	5.1
RHS	1.5	45.4
Update	0.2	0.2

Intermediate data structures for RHS computation are represented in Structures of Arrays (SoA) format since it is required by some kernels to benefit from vectorization, which in our solver is implemented with SSE and AVX instructions. The performance is further increased by the use of division and square root operations which are less accurate (1.5/2.5 ulps) [18] for the computation of the WENO values.

In order to increase the *Arithmetic Intensities* (AI), i.e. ratios of FLOP over total amount of bytes transferred, the AoS/SoA conversion is "fused" with the conversion of flow quantities from conserved to primitive (Figure 1). Similarly, copying back the RHS from SoA to AoS is fused with the first stage of the low-storage Runge-Kutta time stepper. We further replace the conditional branches with conditional moves in the HLLE fluxes.

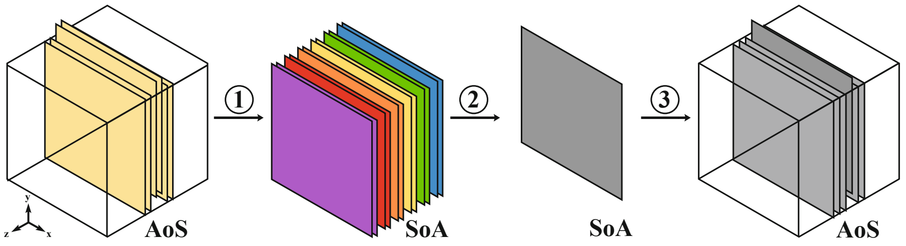

Fig. 1. Slice-wise processing of one block with ring buffers made of two slices (left to right). The two slices undergo a conversion stage (denoted as stage 1) from AoS to SoA format (colors differentiate the 6 flow quantities). This data is then processed by the computing kernels (denoted as stage 2) and the computed RHS (blue) is written back (stage 3) to the associated slice (gray) in the block.

The core layer is composed of three kernels: the RHS, the dt (which computes the time step based on the maximum characteristic velocity) and the update (which updates the conserved quantities) kernels.

The respectives AIs and OIs for the three kernels considered in this work are shown in Table 1. The RHS kernel has an OI that is dependent on the block size and is of 45.4 FLOP/B for a block size of 32, meaning that potentially it could reach peak performance. The update and dt kernel, on the other hand, have fixed OIs of 0.17 and 5 FLOP/B respectively. Without the use of blocks, the OI of the RHS kernel decreases 3.6 FLOP/B, whereas the other two kernels are not affected. The AIs for RHS, dt and update are 1.5, 1.7 and 0.17 FLOP/B respectively, which means that they cannot reach peak performance without exploiting caches.

Node Layer. This layer relies on the core layer and runs simulations on a single computing node, i.e. a full multiprocessor node by using the block size as the parallel grain size. The choice of the concurrency level, i.e. the number of logical threads, is not obvious since one of the considered platforms features multiple hardware threads per core. Since computing the RHS in our solver is limited by the memory bandwidth, we choose to have a fully-subscribed node as we expect that the resource contention is hidden by the memory access costs.

Load imbalance in our solver is not as dramatic as that encountered by spatially adaptive solvers, as the number of blocks contained in our grid remains constant in time. It is however an existing concern due to the presence of ghosts in conjunction with the use of NUMA architectures which make the computation asymmetrical with respect to the cores. To mitigate this issue, we employ OpenMP to maximize the thread-level parallelism (TLP), and by way of static scheduling to initialize the data on ccNUMA platforms, which are subject to the "first-touch" policy.

4 Results and Analysis

In the development of our software we target both supercomputing clusters and high-end desktop machines and in this work we consider a platform for each of these two classes.

Intel Sandy Bridge. The high-end desktop system is represented by an Intel Core i7-2600K (released in January 2011), a quad-core processor featuring Intel's Hyper-Threading Technology and running at 3.4 GHz. The processor is based on the Sandy Bridge micro-architecture and supports the AVX instruction set. The machine has 16 GB of DRAM memory, its measured peak performance is 213 GFLOP/s in single precision and its measured peak DRAM bandwidth is 20.5 GB/s.

AMD Magny-Cours. The supercomputing cluster node is an AMD Magny-Cours (released in March 2010), a 4P 12-cores AMD Opteron 6174 based on the Barcelona micro-architecture with 48 cores at 2.2 GHz and support for the SSE instruction set. One socket contains two ccNUMA nodes (exa-cores), each with one dedicated dual-channel memory controller. The node features 16 GB of DRAM memory per socket for a total of 64 GB. The measured peak performance is 842 GFLOP/s in single precision and the measured peak DRAM bandwidth is 96 GB/s.

Computational Settings. We consider a 3D computational domain size of (1,0.5,0.5) for the simulation of a shock-bubble interaction. A helium bubble with an initial radius of 0.05 is centered at (0.15,0.25,0.25) and a shock wave of $M = 3$ is placed at $x = 0.075$ in air. The specific heat ratios of air and helium are $\gamma = 1.4$ and $\gamma = 1.667$ respectively, whereas the helium to air density ratio is set to 0.138.

Figure 2 shows the isosurface of the helium/air interface location at $\tilde{t} = 2$ (left) and the volume rendering of the density field for this simulation at the same time (right). The shock passage compresses the bubble and deposits vorticity on the helium/air interface. The counter-rotating vorticity pair advects the bubble downstream, turns it inwards and forms a primary vortex ring (PVR) and a smaller secondary vortex ring (SVR).

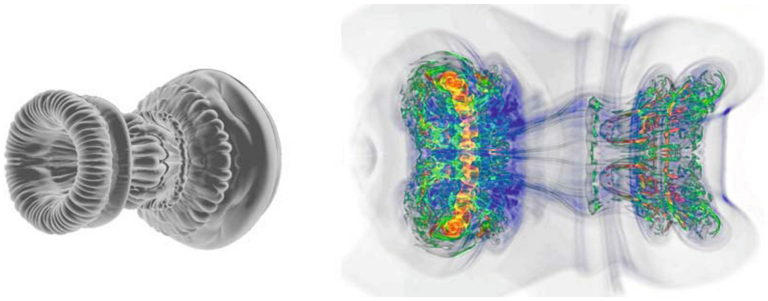

Fig. 2. Shock-Bubble interactions at Mach 3: Isosurface rendering of the interface location (left). Volume rendered image of the vorticity magnitude field for a simulation with 2 billion grid cells (right).

Core Layer. The core layer results presented in the following are obtained by running single precision computations of the same kernels on all cores, in order to capture possible resource contentions. On Magny-Cours, each kernel is explicitly vectorized with SSE whereas on Sandy Bridge AVX is used.

Since the RHS kernel is the most expensive, the studies on the core layer reported are focused on this kernel. We investigate the performance impact of the block size, which represents a tradeoff between spatial locality and memory footprint of the working data set. The left picture of Figure 3 shows the per-core RHS performance versus the block size. We observe that this kernel reaches 35-45% of the peak. Secondly, we note that the performance on Sandy Bridge system is 3-4X than on Magny-Cours, which delineates an important difference between the two platforms: the former's computational power is given by a few cores at high frequency and vector width while the latter enjoys the performance from many cores running at lower frequency. Using a block size of 32 with respect to a block size of 16, we observe an improvement of 4-12% in performance.

With the best block size found in the first performance benchmark, we examine the performance gains provided by the individual optimization techniques applied, namely the use of ring buffers, vectorization and 1.5/2.5 ulps operations. As illustrated in Figure 3 (right), the overall performance gain over the baseline is 14X on Sandy Bridge and 9X on Magny-Cours. The vectorization yields the largest contribution: 9X with AVX and 5X with SSE. Ring buffers contribute another extra 1.2-3.4X. A comparison between the columns for the block-based

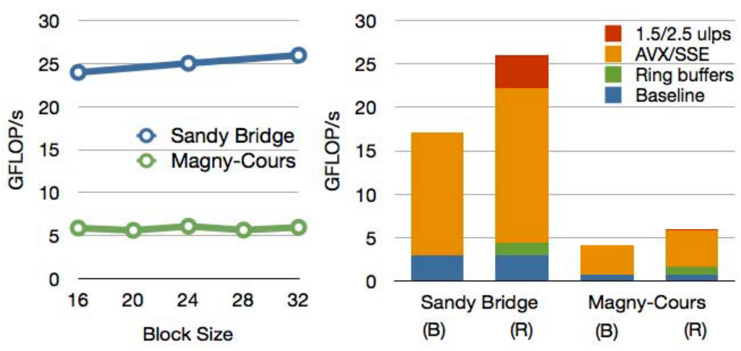

Fig. 3. Performance of the core layer versus block size for the RHS computation (left) and improvements brought by the optimization techniques ('B' indicates block-based memory layout and 'R' stands for ring buffers) (right).

memory layout and the ring buffers further shows that ring buffers also positively affect the performance gain provided by vectorization. We observe that the 1.5/2.5 ulps operations on Sandy Bridge provide a further improvement of 2.3X, whereas for the Magny-Cours platform the gain is only 30%.

We further assess the performance of the kernels by comparing it with the range of the achievable performance estimated with the roofline model. For each kernel, the maximum is computed with its OI (see Table 1), which in our case represents an optimal use of caches, whereas the lower bound is estimated by its AI (see Table 1), which represents a case where the memory hierarchy is not exploited.

As shown in Table 2, we clearly see that, for all kernels, the measured performance is within the estimated intervals. The column "efficiency" denotes the fraction of the peak achievable performance reached by the kernel given its OI, whereas "HU", which stands for hardware utilization, represents the fraction of absolute peak performance.

The RHS and dt kernels reach 48% and 42% of the peak on the Sandy Bridge, with the former kernel being compute bound and the latter being memory bound. Due to its low OI, the update kernels is strongly bound by memory bandwidth and only reaches 2% of the peak performance, although it shows a 100% efficiency. However, because of its simplicity and its total lack of temporal locality, the update kernel has no space for improvement.

Node Layer. We investigate the additional costs incurred by the necessary copying of ghost values between blocks, by the increased memory footprint required to run a simulation and by potential load imbalance issues.

We consider a weak scaling benchmark in terms of GFLOP/s for a system size of $512 \times 256 \times 256$, shown in Figure 4. On the Magny-Cours, the thread placement scheme for the benchmark was chosen so that NUMA nodes are filled first and thread-data affinity is maintained. This causes the piecewise behavior of the (OI-based) predicted performance shown by the solid line: each step represents the

Table 2. Measured performance of the core layer and predicted range, efficiency w.r.t. the maximum achievable performance, nominal peak and fraction of the peak (HU)

Kernel	Platform	Measured [GFLOP/s]	Range [GFLOP/s]	Efficiency [-]	Peak Perf. [GFLOP/s]	HU [-]
RHS	Sandy Bridge	26.0	7.8 - 54.4	48%	54.4	48%
	Magny-Cours	6	3 - 17.6	34%	17.6	34%
update	Sandy Bridge	0.9	0.9 - 0.9	100%	54.4	2%
	Magny-Cours	0.3	0.3 - 0.3	100%	17.6	2%
dt	Sandy Bridge	22.8	8.7 - 26	88%	54.4	42%
	Magny-Cours	4.6	3.4 - 10.2	45%	17.6	26%

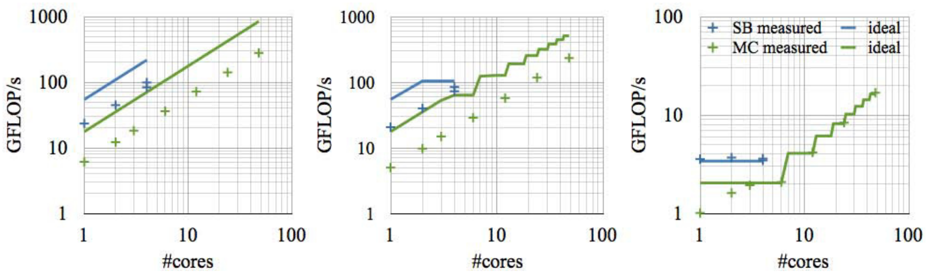

Fig. 4. Weak scaling plot for the RHS (left), dt (center) and update (right) kernels

addition of a NUMA node, and therefore an increase in bandwidth available, to the group of nodes on which the test runs.

On Sandy Bridge we observe speedups up to 3.7X (over 4 cores), whereas on Magny-Cours the speedup reaches 45X (over 48 cores). As illustrated in Figure 4 (left), on Magny-Cours the RHS performance scales almost linearly, attaining 246 GFLOP/s. On Sandy Bridge we observe similar results, with the addition of a point representing the performance obtained by running two hardware threads per core, which increases the performance from 80 GFLOP/s to 92 GFLOP/s, leading to 42% of the peak. A similar behavior is observed for the dt kernel (Figure 4, middle). On Sandy Bridge, the kernel reaches 81 GFLOP/s, corresponding to 37% of the peak, whereas for the same kernel Magny-Cours yields 362 GFLOP/s, corresponding to 43% of the peak performance. We note that the expected peak performance on the two platforms changes from compute bound to memory bound as the number of threads increases, due to contention of the memory bandwidth which decreases the resources available per core.

Figure 4 (right) shows the measured performance for the update kernel. On Sandy Bridge the bandwidth is already saturated with a single thread and the performance reached is around 3.5 GFLOP/s. With two threads the performance of the kernel is slightly above the prediction: we attribute this to caching effects in the LLC. On Magny-Cours the measured performance is 2.3 GFLOP/s with one NUMA node and 16 GFLOP/s with 8 nodes.

For the considered simulation of SBI, the time spent on Magny-Cours for a single time step is measured to be 2.9 seconds with 98% of the time spent in the RHS kernel and 1% in the update and the dt kernels, which further justifies the effort spent on the RHS kernel.

5 Conclusions and Outlook

We have developed efficient CPU kernels for compressible multiphase flows and performed an a-priori analysis to estimate the maximum achievable performance using the roofline model. We have presented a high performance uniform resolution simulation software for multiphase compressible flows tailored to state-of-the-art multicores, capable of reaching up to 48% of the peak performance on the considered platforms. To the best of our knowledge, this is the highest fraction of the peak performance obtained for a compressible flow simulation software on shared memory architectures. These results outperform by 9-14X the ones considered to be state-of-the-art.

On the core layer, the improvement brought by the optimization techniques have provided a performance gain of one order of magnitude over the baseline C++ implementation. Furthermore, all of our measurements are within the performance bounds predicted by the roofline model.

On Magny-Cours, we have reported a speedup of 40-45X over 48 cores, reaching 360 GFLOP/s which corresponds to 42% of the peak performance and of 3.7X over 4 cores on a single-socket Sandy Bridge platform.

Because of the block-based nature of the present solver, the optimizations considered here are also applicable to other frameworks such as AMR. These are currently being applied to wavelet-based adaptive solvers [4] for compressible flow simulations with and without solid boundaries.

Acknowledgments. The authors wish to thank Dr. Olivier Byrde, Teodoro Brasacchio and Adrian Ulrich of the Brutus cluster support team at ETH Zurich for their crucial assistance.

References

1. Quirk, J., Karni, S.: On the dynamics of a shock-bubble interaction. Journal of Fluid Mechanics 318, 129–163 (1996)
2. Colella, P., Graves, D.T., Ligocki, T.J., Martin, D.F., Mondiano, D., Serafini, D.B., Van Straalen, B.: Chombo software package for amr applications design document. Technical report, Lawrence Berkeley National Laboratory (2003)
3. Alam, J.M., Kevlahan, N.K.R., Vasilyev, O.V.: Simultaneous space-time adaptive wavelet solution of nonlinear parabolic differential equations. Journal of Computational Physics 214(2), 829–857 (2006)
4. Rossinelli, D., Hejazialhosseini, B., Spampinato, D., Koumoutsakos, P.: Multicore/Multi-GPU Accelerated Simulations of Multiphase Compressible Flows Using Wavelet Adapted Grids. SIAM J. Scientific Computing 33(2) (2011)

5. Cameron, K., Ge, R., Feng, X.: High-performance, power-aware distributed computing for scientific applications. Computer 38(11), 40–47 (2005)
6. Luk, C.K., Newton, R., Hasenplaugh, W., Hampton, M., Lowney, G.: A Synergetic Approach to Throughput Computing on x86-Based Multicore Desktops. IEEE Softw. 28, 39–50 (2011)
7. Puschel, M., Moura, J., Johnson, J., Padua, D., Veloso, M., Singer, B., Xiong, J., Franchetti, F., Gacic, A., Voronenko, Y., Chen, K., Johnson, R., Rizzolo, N.: SPIRAL: Code Generation for DSP Transforms. Proceedings of the IEEE 93(2), 232–275 (2005)
8. Shalf, J., Quinlan, D., Janssen, C.: Rethinking hardware-software codesign for exascale systems. IEEE Computer 44(11), 22–30 (2011)
9. Chen, G., Chacón, L., Barnes, D.C.: An efficient mixed-precision, hybrid CPU-GPU implementation of a fully implicit particle-in-cell algorithm. ArXiv (2011)
10. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM 52, 65–76 (2009)
11. Petitet, A., Whaley, R.C., Dongarra, J., Cleary, A.: HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers
12. Yelick, K., Semenzato, L., Pike, G., Miyamoto, C., Liblit, B., Krishnamurthy, A., Hilfinger, P., Graham, S., Gay, D., Colella, P., Aiken, A.: Titanium: a high-performance Java dialect. CCPE 10(11-13), 825–836 (1998)
13. Van Straalen, B., Shalf, J., Ligocki, T., Keen, N., Yang, W.S.: Scalability challenges for massively parallel amr applications. In: IEEE International Symposium on Parallel Distributed Processing, pp. 1–12 (2009)
14. Prosperetti, A., Tryggvason, G. (eds.): Computational Methods for Multiphase Flow. Cambridge University Press, Cambridge (2007)
15. Jiang, G., Shu, C.: Efficient implementation of weighted ENO schemes. Journal of Computational Physics 126(1), 202–228 (1996)
16. Wendroff, B.: Approximate Riemann solvers, Godunov schemes and contact discontinuities. In: Toro, E.F. (ed.) Godunov Methods: Theory and Applications, pp. 1023–1056. Kluwer Academic/Plenum Publ. (2001)
17. 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, pp. 4:1–4:12. IEEE Press (2008)
18. AMD Inc.: Software Optimization Guide for the AMD 15h Family (2011)