





Article

Optimization of finite-differencing kernels for numerical relativity applications

Roberto Alfieri^{1,2}  0000-0002-6201-9335, Sebastiano Bernuzzi^{1,2}  0000-0002-2334-0935, Albino Perego^{1,2,3}  0000-0002-0936-8237, David Radice^{4,5}  0000-0001-6982-1008

¹ Dipartimento di Scienze Matematiche Fisiche ed Informatiche, Università di Parma, I-43124 Parma, Italy

² Istituto Nazionale di Fisica Nucleare, Sezione Milano Bicocca, gruppo collegato di Parma, I-43124 Parma, Italy

³ Istituto Nazionale di Fisica Nucleare, Sezione Milano Bicocca, I-20126 Milano, Italy

⁴ Institute for Advanced Study, 1 Einstein Drive, Princeton, NJ 08540, USA

⁵ Department of Astrophysical Sciences, Princeton University, 4 Ivy Lane, Princeton, NJ 08544, USA

* Correspondence: sebastiano.bernuzzi@gmail.com

Abstract: A simple optimization strategy for the computation of 3D finite-differencing kernels on many-cores architectures is proposed. The 3D finite-differencing computation is split direction-by-direction and exploits two level of parallelism: in-core vectorization and multi-threads shared-memory parallelization. The main application of this method is to accelerate the high-order stencil computations in numerical relativity codes.

Keywords: Numerical Relativity; Many-core architectures; Knight Landing; Vectorization

1. Introduction

Numerical relativity (NR) is the art of solving Einstein's equations of general relativity using computational physics techniques. The typical NR application is the simulation of strong-gravity astrophysical phenomena like the collision of two black holes or two neutron stars, and the collapse/explosion of massive stars. Those simulations are of primary importance for fundamental physics and high-energy astrophysics, including the emerging field of gravitational-wave and multi-messenger astronomy [1–3]. Some of the open problems in the simulation of strongly gravitating astrophysical sources demand the modeling of the dynamical interaction between supranuclear-density, high-temperature matter [4], and neutrino radiation in complex geometries [5]; the resolution of magnetohydrodynamical instabilities in global simulations [6,7]; the production of accurate gravitational waveforms for many different physical settings [8,9].

Addressing these challenges will require significant computational resources and codes able to efficiently use them. For example, in the case of neutron star mergers, typical production simulations use few hundreds cores and require $\sim 100,000$ CPU-hrs to cover the dynamical timescale of the last orbits ($\mathcal{O}(100)$ milliseconds of evolution). However, to fully capture hydromagnetic instabilities in the merger remnant, necessary to predict the electromagnetic signature of mergers, it would be necessary to increase the resolution in these simulations by more than a factor ten. With current technologies, a simulation at this scale would require one billion CPU-hrs. Clearly, exascale resources are needed to perform these simulations.

Exascale high-performance computing, based on energy-efficient and heterogeneous architectures, offers the possibility to tackle some of scientific challenges of relativistic astrophysics in strong-gravity regime. On the technological side, the Intel Knight Landing (KNL) processor brings up most of

the features required for the upcoming exascale computing, such as power efficiency, with a large FLOPs/watt ratios, and high peak performance, provided by the many-core architecture and by large vector instructions [10]. The KNL many-core architecture allows users to approach exascale systems with the standard MPI/OpenMP programming model that is standard for NR codes. More recently, Intel released the Skylake (SKL) processor microarchitecture that succeeds the Broadwell (BDW) and supports the same AVX-512 instruction set extensions as the KNL. Vectorization has a key role since it introduces an increasing speed-up in peak performance, but existing NR codes do require refactoring or even the introduction of new programming paradigms and new algorithmic strategies for the solution of Einstein equations.

This work discusses the first steps towards the implementation of a highly scalable code that will be dedicated to NR simulations for GW astronomy and relativistic astrophysics. The primary focus is on the vectorization of the basic kernels employed for the discretization of the spacetime's metric fields. We consider the optimization of the computation of finite-differencing derivatives on a spatially uniform, logically Cartesian patch using high-order-accurate stencils in 3D. The problem has been discussed rarely since high-order finite-differencing operators are peculiar to NR applications, e.g. [11–14], but not often employed in other branches of relativistic astrophysics (but see [15,16] for an optimization discussion). Here, we propose an optimization strategy based on the use of OpenMP 4.0 PRAGMA and on two level of parallelism: in-core vectorization and multi-threads shared memory parallelization.

2. Method

The general relativistic description of the spacetime (metric fields) plus matter and radiation fields reduces to solving an initial-boundary value problem with nonlinear hyperbolic partial differential equations (PDE) in 3 spatial dimensions plus time (3+1 D). A toy model equation for the metric fields is the tensor wave equation,

$$\partial_t h_{ij} = -2K_{ij} \quad (1a)$$

$$\partial_t K_{ij} = R_{ij}, \quad (1b)$$

obtained by linearizing Einstein equation around a reference background, $\mathbf{g} = \boldsymbol{\eta} + \mathbf{h}$. Above, $i, j = 1, 2, 3$ are spatial indexes, h_{ij} is the perturbed metric, K_{ij} the extrinsic curvature, and the Ricci tensor takes the form

$$R_{ij} = -\frac{1}{2}\eta^{kl}\partial_k\partial_l h_{ij}, \quad (1c)$$

where η^{ij} is the inverse background metric and a sum on k is understood (Einstein's summation convention). An even simpler toy model retaining all the key features is the 3D scalar wave equation,

$$\partial_t \phi = \Phi, \quad (2a)$$

$$\partial_t \Phi = \Delta \phi = \eta^{ij}\partial_i\partial_j \phi. \quad (2b)$$

The PDE system in Eq. (1) is in a first-order-in-time and second-order-in-space form and mimics the conformal BSSNOK and Z4 free evolution schemes for general relativity [17–20]. Note the basic operations of the RHS of Eq. (1) are derivatives, metric inversion and contractions (multiplication, divisions and sums).

The numerical solution of Eq. (1) and Eq. (2) is based on finite differencing. The 3D flat space is discretized with a uniform Cartesian mesh of grid spacing $h_x = h_y = h_z = h$ and of size $n^3 = n_x n_y n_z$;

Table 1. Coefficients of 1D finite differencing stencils for first (top) and second (bottom) derivative. Stencils are symmetric.

S	0	1	2	3	4
2	0	2/3	-1/12		
3	0	3/4	-3/20	1/60	
4	0	4/5	-1/5	4/105	-1/280
2	-5/2	4/3	-1/12		
3	-49/18	3/2	-3/20	1/90	
4	-205/72	8/5	-1/5	8/315	-1/560

each component of the fields on a given grid-node is $u_{i,j,k} = u(x_i, y_j, z_k)$. The derivatives in the RHS, e.g., in the y -direction, are approximated by

$$(\partial_y u)_{i,j,k} \approx h^{-1} \sum_{s=-S}^S c_s u_{i,j+s,k}, \quad (3)$$

where S is the size of the stencil and the error term scales as $\mathcal{O}(h^{2S})$ (See Tab. 1). Similar expressions hold for the other directions. Mixed derivatives are approximated by the successive application of 1D finite differencing operators. The method of lines and standard explicit integration algorithms are then used to time-update the state vector $\mathbf{u}_{i,j,k} = \{u_{i,j,k}\}$ composed of all the field components with the equations in the form $\dot{\mathbf{u}}_{i,j,k}(t) = F_{i,j,k}(\mathbf{u}, \partial\mathbf{u}, \partial\partial\mathbf{u})$. Our work focuses on the optimization of the finite differencing kernel and tensor contractions for the computation of the RHS.

Our global strategy for the RHS computation is composed of two levels: (i) in-core vectorization, that takes advantage of single instruction multiple data (SIMD) parallelism on aligned memory; and (ii) multi-thread and shared-memory parallelism of the outer loops. (ii) splits RHS evaluations in block of operations (derivatives, contractions, etc) in order to fit instructions and data cache. We expect the latter optimization to be particularly important for the Einstein equations, where the RHS evaluation involve accessing tens of grid functions.

The data $u_{i,j,k}$ is stored in contiguous arrays of floats or double. The x -direction is the only one where elements are contiguous in memory; thus SIMD parallelization is only possible along its fastest running index. The other two directions are accessed as $u[\text{ijk} + s \cdot dj]$ or $u[\text{ijk} + s \cdot dk]$, where $dj = n_x$ and $dk = n_x n_y$. So for the approximation of the derivatives in the y and z directions strided memory access is unavoidable. Instead of using vector instructions to evaluate (3), we vectorize the code by grouping together points in the x -direction, i.e., the derivative in the y -direction is computed as

$$(\partial_y u)_{i,j,k} \approx \sum_{s=-S}^S c_s u_{i,j+s,k}, \quad (4a)$$

$$(\partial_y u)_{i+1,j,k} \approx \sum_{s=-S}^S c_s u_{i+1,j+s,k}, \quad (4b)$$

⋮

$$(\partial_y u)_{i+V,j,k} \approx \sum_{s=-S}^S c_s u_{i+V,j+s,k}, \quad (4c)$$

where V is the size of the vector register, e.g., 8 on KNL when computing in double precision. This simple in-core parallelization strategy is then combined with threading for out-of-core, but in-node parallelism using the OpenMP library. Note that our implementation requires the additional storage of 1D arrays with temporary data. However, for typical block sizes (32^3 or larger) the additional memory requirements are negligible, even when storing hundreds of grid functions.

Table 2. BDW, KNL, and SKL characteristics and performance of the nodes used for the tests.

Node type	Intel Xeon	Frequency	Cores	HT	Core/Node perf	cache
BDW	E5-2684 v4	2.3 GHz	2x18	off	36/1300 GFlops	256 KB L2, 40 MB Smart Cache
KNL	Phi 7250	1.4 GHz	1x68	on	44/3000 GFlops	1 MB L2, 16 GB MCDRAM
SKL	8160	2.1 GHz	2x24	off	67/3200 GFlops	1 MB L2, 33 MB L3

A pseudo code of our RHS implementation is the following:

To ensure optimal performance of our code, we perform aligned memory allocation at 64bit boundaries and we ensure that the block size is a multiple of the vector size. This avoids the need for and the use of remainder loops. Note that avoiding reminder loops is also necessary to ensure the exact reproducibility of the calculations. The block size is fixed and set at compilation stage.

3. Experimental setup

The above method is tested on BDW, KNL and SKL nodes of the Marconi cluster at CINECA, and on BDW and KNL nodes of the HPC data center at the University of Parma. The characteristics of the nodes are listed in Tab. 2. ¹

We consider independent implementations of the scalar wave and linearized Einstein equations and exploit the auto-vectorization capabilities of the Intel C/C++ compiler combined with the introduction of the PRAGMA SIMD statement. This approach strikes a balance between performance and code portability. The specific options for auto-vectorization that we employed are:

```
icc -O3 -xCORE-AVX2 -qopt-report=4 # do Vectorization (V) on BDW
icc -O3 -xMIC-AVX512 -qopt-report=4 # do Vectorization (V) on KNL
icc -O3 -xCORE-AVX512 -qopt-report=4 # do Vectorization (V) on SKL
icc -O3 -no-vec -no-simd # do Not Vectorize (NV)
```

Note that, in order to completely disable vectorization, we use the options -no-vec for compiler auto-vectorization and -no-simd to disable the PRAGMA SIMD statements. Double precision data type and align arrays with 64 bytes-wise vector are employed in all our tests.

4. Results

4.1. Wave equation

We explore the performances of our method on BDW, KNL and SKL using the wave equation implementation. Comparative tests on those architectures on a single core/thread are first considered for variable block size up to 128 points, and with focus on vectorization performances. Multi-thread performances and strong OpenMP scaling are then analyzed with a fixed block size of 128 points. The speed of our implementation is measured in terms of Million cells updates per second; figures refer to the speed of the whole program that is dominated by the RHS evaluation.

The results relative to the single core optimization are shown in Fig. 1. Without vectorization the speed on the KNL core is about a factor 3 smaller than BDW; SKL is about 20% faster then BDW. When vectorization is enabled, we find a speedup of of 1.5 on BDW, of 4 on KNL, and of 2 on SKL. This in-core optimization results are about half the theoretical maximal speedup of a single core due to the vectorization: 8x for KNL/SKL and 4x for BDW.

¹ The peak performance reported in the Table are calculated assuming 2 instructions per cycle (superscalar processors), purely fused multiply-add (FMA) instructions and full SIMD (see [21]). So for example the theoretical peak performance on XEON Phi 7250 is: $2(\text{superscalar}) \times 2(\text{FMA}) \times 8(\text{SIMD}) = 32\text{Flops/cycle DP}$ per core with vectorization enabled and $2(\text{superscalar}) \times 2(\text{FMA}) = 4\text{Flops/cycle DP}$ per core without vectorization.

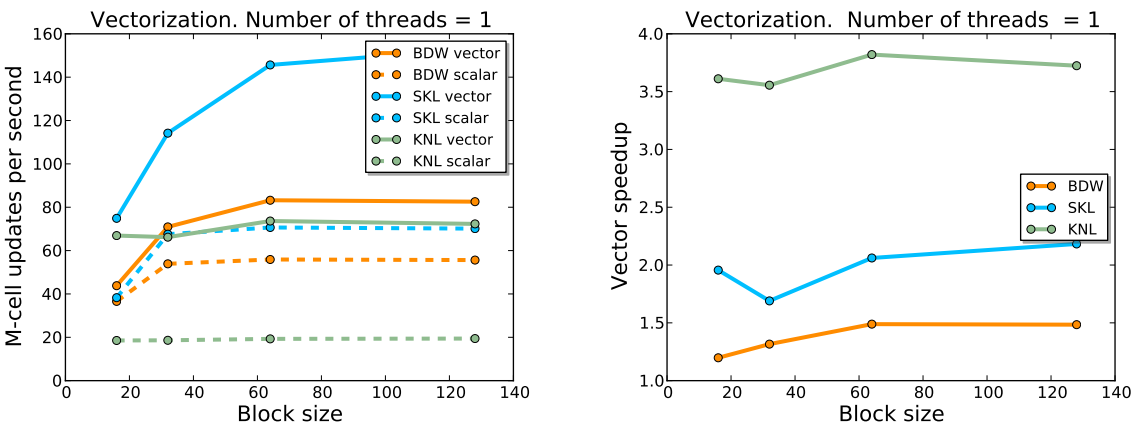


Figure 1. Single-core performances (left panel) and speedup (right panel) for variable block size in the case of the wave equations with stencil size $S = 2$. The tests have been executed with vectorization enabled (solid lines) and disabled (dashed lines). The best performance is obtained for vectorized kernel on SKL.

Table 3. Vectorization speedup on single core for the wave equation with stencil size $S = 2$. The block size is 128^3 . The table shows the Intel compiler report information (obtained with the `-qopt-report` options) and the measured speedup (ratio between non-vectorized and vectorized execution time). The measured speed-up differs of about a factor 2 or more from the potential speedup.

	BDW		KNL		SKL	
Operation	Compiler	Measure	Compiler	Measure	Compiler	Measure
Derivative	5.03	1.7	6.58	3.7	5.73	2.22
Contraction	5.61	1.8	7.77	4	5.61	2.27

Table 3 reports speedup measurements on the two main operations of the RHS (contraction and derivatives). Such numbers differs of about a factor 2 or more from the potential speedup reported by the compiler at compilation time (also reported on the table). Performances are stable for variable block size as soon as the number of floating point operations dominate the computation ($n \gtrsim 32$).

All data fits the cache memory on the considered systems (see also below), so the sub-optimal performance of our implementation can not be ascribed to the memory speed. Our results are in-line with those reported by Intel in a similar test [16].

The results relative to the multi-thread optimization are shown in Fig. 2. The grid size employed for these test is $n = 128$. The latter block size correspond to a grid with a maximum memory occupation of 33 MBytes. BDW smart cache, L3 SKL cache and KNL MCDRAM are large enough for the considered block sizes, so the memory speed does not represent a bottleneck even for the single thread computation. In KNL systems the MCDRAM High Bandwidth Memory is 4 times faster than DDR4. The MCDRAM can be used as a cache memory if the node is configured in Cache Mode. If the node is in Flat Mode it is possible to allocate data on MCDRAM through the High-Bandwidth Memory (HBM) library or through the memory affinity control provided by the `numactl` tool. We tested both methods obtaining the same results.

Scaling on BDW and SKL is close to ideal until 16 threads, then we observe a drop of the performance even running on 32 physical cores. On the other hand the KNL shows a sustained speed-up (about 4x) up to 64 physical cores, and remains good with hyper-threading. The use of the KNL node can, in principle, speed-up computations of more than a factor of 2 with respect to BWD and SKL when the kernel works on sufficiently large block sizes. Note, however, the use of $n = 128$ is unrealistic for numerical relativity application since one would be memory limited in that case.

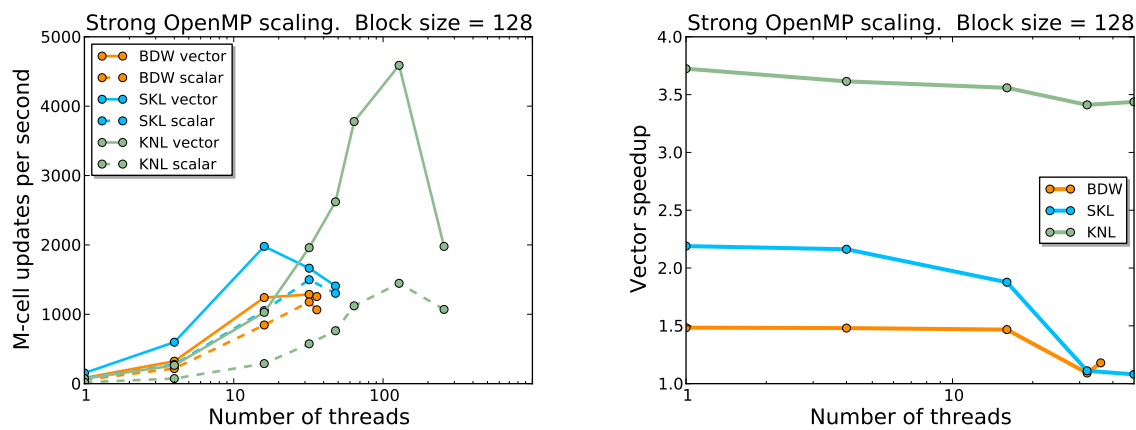


Figure 2. Multi-thread performances (left panel) and speedup (right panel) for block size $n = 128$ as a function of the number of threads, in the case of the wave equations with stencil size $S = 2$. KNL demonstrates good speedup, overall performance and scalability, especially in case of large block sizes.

4.2. Linearized Einstein equations

Let us now discuss the results on linearized Einstein equations.

Besides stencil operations the numerical solution of Einstein's equations also requires the evaluation of many tensor operations, such as the contraction of tensor indices and the inversion of small 3×3 or 4×4 matrices. Since these operations are local to each grid point and involve the multiple re-use of data already in cache, they can be more efficiently vectorized. The linearized Einstein equations are not as algebraically complex than their fully non-linear counterpart, but can give an indication of the speedup that could be achieved with vectorization in a production simulation.

We perform a study of the performance of our vectorization and threading strategies for the linearized Einstein equations using the same compiler options as in Sec. 4.1. The tests are performed using a single KNL node on CINECA Marconi. Our results are summarized in Figs. 3 and 4.

The single core performances are very encouraging, especially for large block sizes. Indeed, on a single core, the vectorized code can achieve close to a factor ~ 8 speedup over the non-vector version of our code. This is the theoretical maximum speedup for double precision calculations if fused math-addition operations are not used. Even for smaller block sizes, down to $N = 8$, we achieve a speedup factors of ~ 5 – 6 . This shows that vectorization would be very beneficial also for production simulations, where the block sizes are typically $n = 32$ or smaller.

When multithreading, the speedup due to vectorization is somewhat worse. On the one hand, this is expected because each thread operates on a smaller sub-block of the data. On the other hand, when comparing vector speedups with threading to those obtained in an equivalently smaller single core case, we typically find worse speedups in the former case (see Fig. 4). This might indicate that the default tiling employed by OpenMP is not efficient. Improvements could be obtained either by tuning the tiling, or by switching to a coarse grained parallelization strategy also for OpenMP. The latter could be, for example, to map threads to individual (small) blocks.

We extended our OpenMP scaling tests all the way up to 68 cores, thus using all of the physical cores on a KNL node. However, we find the results with 68 threads to be somewhat inconsistent between runs: sometimes running on 68 threads yields a speedup and sometimes a slowdown. We find that leaving 4 physical cores dedicated to OS and IO tasks result in more predictable performances. Consequently, we do not plan to perform simulations using more than 64 OpenMP threads on the KNL architecture. Nevertheless, in Figs. 3 and 4 we show the results obtained with 68 threads for completeness.

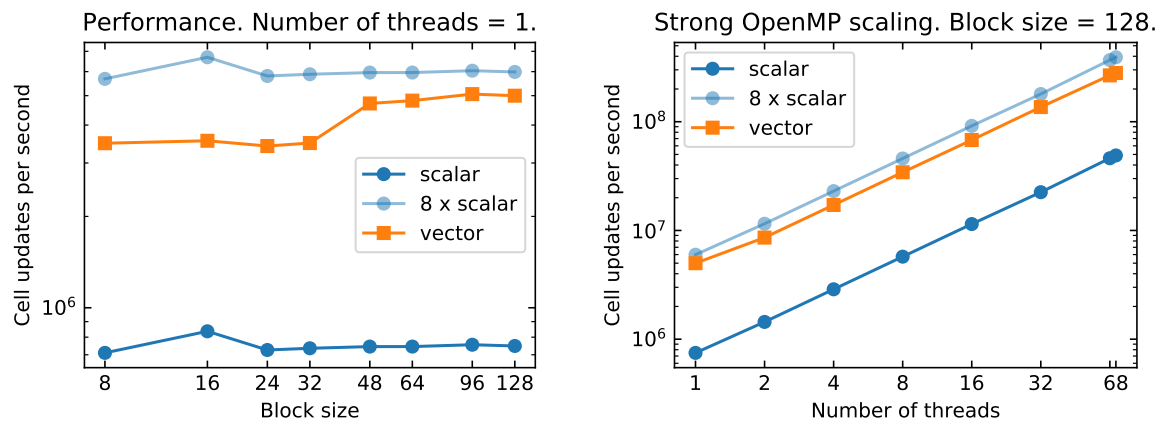
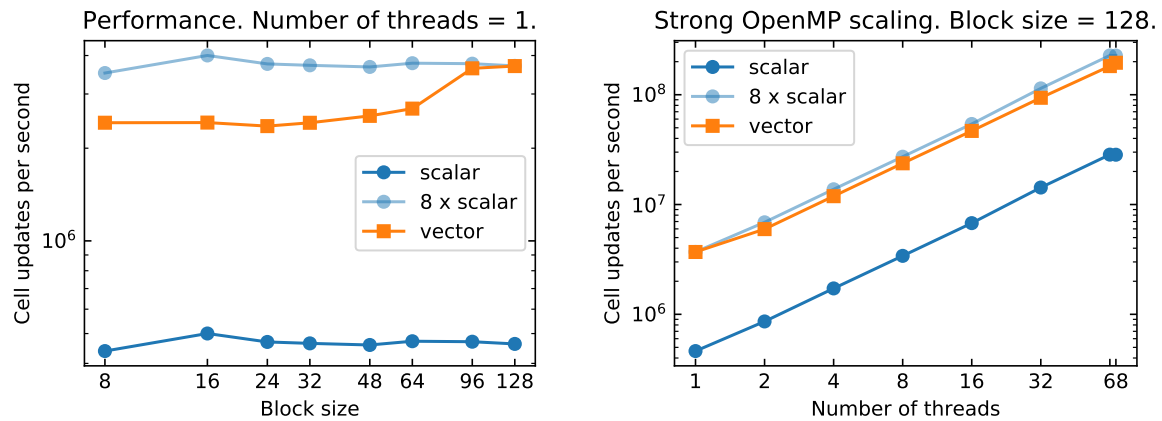
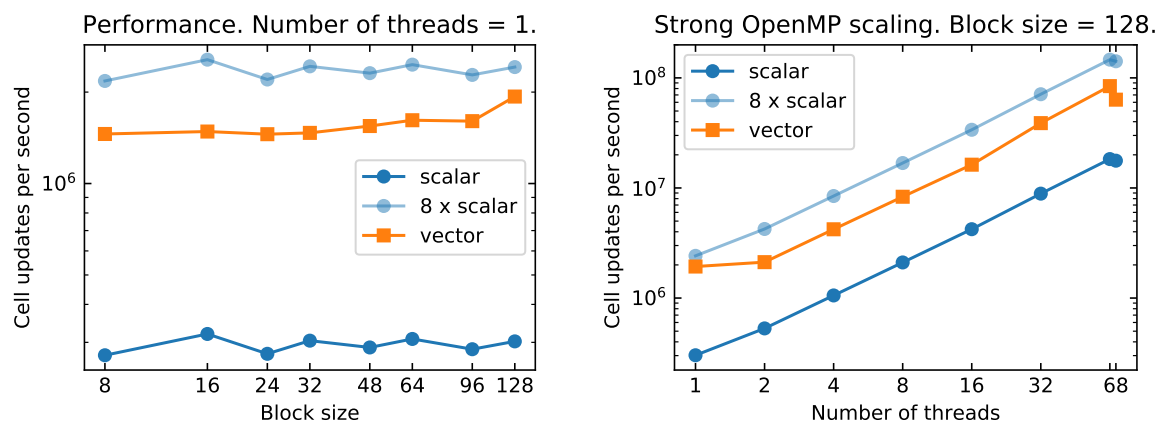
Case $S = 2$.Case $S = 3$.Case $S = 4$.

Figure 3. Single-core performances, as a function of the block size (left panels), and strong scaling with OpenMP, as a function of the number of threads (right panels), for the linearized Einstein equations with different stencil size (top: $S = 2$, middle: $S = 3$, bottom: $S = 4$). We find nearly ideal vector speedup and scaling for large block sizes. However, the code performances appear inconsistent when using all of the 68 physical cores on the node, possibly because of the effect of system interrupts.

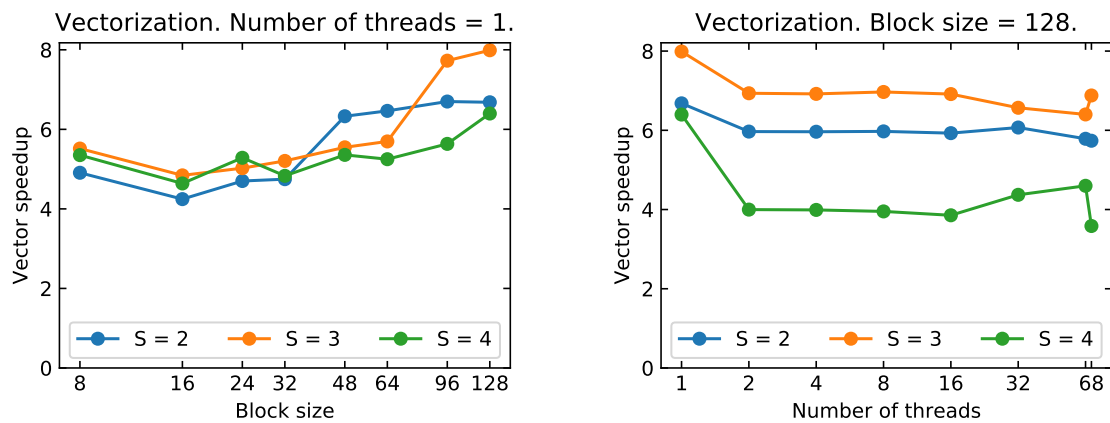


Figure 4. Vectorization speedup for $S = 2$, $S = 3$, and $S = 4$. Left panel: single core vector speedup. Right panel: vector speedup for increasing thread count. Good vector efficiency is achieved for large block sizes, even though the speedup due to vectorization shows an unclear trend with S . The results when using 68 threads might be affected by system interrupts.

5. Conclusion

Motivated by the future need of developing a highly scalable code for NR application on exascale computing resources, we have introduced and tested an optimization strategy to calculate 3D finite-differencing kernels on different many-core architectures, including BDW, KNL and SKL. The proposed method can be implemented with a minimal programming effort on existing NR codes. It gives substantial speed-up of both contraction and 3D stencil computations on BDW, KNL and SKL architectures.

A deeper code re-factory would be needed in order to further improve performances. For example the introduction of the loop tiling technique would guarantee a better exploitation of the L1 and L2 caches. Using intrinsic instructions can further improve the performance. This approach would limit our possibility to keep the code simple and portable but we plan to test it in a compute intensive loop of our kernel in order to assess the further speed-up.

We think compilers can heavily improve their auto-vectorization capabilities and we expect the Intel compiler to continue to improve in upcoming releases, as partially confirmed by similar conclusion reported in [16]. In our early work we also tested the GNU compiler (v. 5.4.0), but we had to move to the Intel compiler since the vectorization support did not demonstrate to be ready for the production stage. Work on GNU compilers will continue using the next compiler releases for code portability reasons.

Future work will be focused on improving the multi-thread performances of our approach and on embedding it into a distributed memory and adaptive grid framework.

Acknowledgments: S.B. acknowledges support by the European Union's H2020 under ERC Starting Grant, grant agreement no. BinGraSp-714626. DR acknowledges support from a Frank and Peggy Taplin Membership at the Institute for Advanced Study and the Max-Planck/Princeton Center (MPPC) for Plasma Physics (NSF PHY-1523261). Computations were performed on Marconi/CINECA (PRACE proposal 2016153522 and ISCRA-B project number HP10B2PL6K), and on the HPC facility of the University of Parma, Italy.

Author Contributions: S.B. and D.R. conceived and designed the project; A.R. performed the analysis of the wave equation code at CINECA and Parma; S.B. implemented the wave equation code; D.R. implemented the linearized Einstein code and performed the the analysis of the linearized Einstein code at CINECA; A.P. performed the analysis of the wave equation code at CINECA; All authors contributed to discussion and paper writing.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BDW	Intel Broadwell architecture
FMA	Fused multiply-add
KNL	Intel Knight Landing architecture
HNM	High-Bandwidth Memory library
MCDRAM	Multi-Channel Dynamic Random-Access Memory
MPI	Message Passing Interface
NR	Numerical Relativity
OpenMP	Open Multiprocessing
RHS	Righ-hand-side
SIMD	Single Instruction Multiple Data
SKL	Intel Skylake architecture

References

- Abbott, B.P.; others. Observation of Gravitational Waves from a Binary Black Hole Merger. *Phys. Rev. Lett.* **2016**, *116*, 061102, [arXiv:gr-qc/1602.03837].
- Abbott, B.P.; others. GW170817: Observation of Gravitational Waves from a Binary Neutron Star Inspiral. *Phys. Rev. Lett.* **2017**, *119*, 161101, [arXiv:gr-qc/1710.05832].
- Multi-messenger Observations of a Binary Neutron Star Merger. *Astrophys. J.* **2017**, *848*, L12, [arXiv:astro-ph.HE/1710.05833].
- Radice, D.; Bernuzzi, S.; Del Pozzo, W.; Roberts, L.F.; Ott, C.D. Probing Extreme-Density Matter with Gravitational Wave Observations of Binary Neutron Star Merger Remnants. *Astrophys. J.* **2017**, *842*, L10, [arXiv:astro-ph.HE/1612.06429].
- Perego, A.; Rosswog, S.; Cabezón, R.; Korobkin, O.; Kaeppli, R.; others. Neutrino-driven winds from neutron star merger remnants. *Mon.Not.Roy.Astron.Soc.* **2014**, *443*, 3134, [arXiv:astro-ph.HE/1405.6730].
- Radice, D. General-Relativistic Large-Eddy Simulations of Binary Neutron Star Mergers. *Astrophys. J.* **2017**, *838*, L2, [arXiv:astro-ph.HE/1703.02046].
- Kiuchi, K.; Kyutoku, K.; Sekiguchi, Y.; Shibata, M. Global simulations of strongly magnetized remnant massive neutron stars formed in binary neutron star mergers **2017**. [arXiv:astro-ph.HE/1710.01311].
- Bernuzzi, S.; Nagar, A.; Thierfelder, M.; Brügmann, B. Tidal effects in binary neutron star coalescence. *Phys.Rev.* **2012**, *D86*, 044030, [arXiv:gr-qc/1205.3403].
- Bernuzzi, S.; Nagar, A.; Dietrich, T.; Damour, T. Modeling the Dynamics of Tidally Interacting Binary Neutron Stars up to the Merger. *Phys.Rev.Lett.* **2015**, *114*, 161103, [arXiv:gr-qc/1412.4553].
- 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* **2016**, *36*, 34–46.
- Brügmann, B.; Gonzalez, J.A.; Hannam, M.; Husa, S.; Sperhake, U.; others. Calibration of Moving Puncture Simulations. *Phys.Rev.* **2008**, *D77*, 024027, [arXiv:gr-qc/gr-qc/0610128].
- Husa, S.; González, J.A.; Hannam, M.; Brügmann, B.; Sperhake, U. Reducing phase error in long numerical binary black hole evolutions with sixth order finite differencing. *Class. Quant. Grav.* **2008**, *25*, 105006, [arXiv:gr-qc/0706.0740].
- Radice, D.; Rezzolla, L.; Galeazzi, F. Beyond second-order convergence in simulations of binary neutron stars in full general-relativity. *Mon.Not.Roy.Astron.Soc.* **2014**, *437*, L46–L50, [arXiv:gr-qc/1306.6052].
- Bernuzzi, S.; Dietrich, T. Gravitational waveforms from binary neutron star mergers with high-order weighted-essentially-nonoscillatory schemes in numerical relativity. *Phys. Rev.* **2016**, *D94*, 064062, [arXiv:gr-qc/1604.07999].
- Borges, L.; (Intel), P.T. 3D Finite Differences on Multi-core Processors.
- Andreolli, C. Eight Optimizations for 3-Dimensional Finite Difference (3DFD) Code with an Isotropic (ISO). Intel Software On-line documentation **2014**.
- Baumgarte, T.W.; Shapiro, S.L. On the numerical integration of Einstein's field equations. *Phys. Rev.* **1999**, *D59*, 024007, [gr-qc/9810065].
- Nakamura, T.; Oohara, K.; Kojima, Y. General Relativistic Collapse to Black Holes and Gravitational Waves from Black Holes. *Prog. Theor. Phys. Suppl.* **1987**, *90*, 1–218.

- 242 19. Shibata, M.; Nakamura, T. Evolution of three-dimensional gravitational waves: Harmonic slicing case.
243 *Phys. Rev.* **1995**, *D52*, 5428–5444.
- 244 20. Bernuzzi, S.; Hilditch, D. Constraint violation in free evolution schemes: comparing BSSNOK with a
245 conformal decomposition of Z4. *Phys. Rev.* **2010**, *D81*, 084003, [[arXiv:gr-qc/0912.2920](#)].
- 246 21. Dolbeau, R. Theoretical peak FLOPS per instruction set: a tutorial. *The Journal of Supercomputing* **2018**,
247 *74*, 1341–1377.