HAL
open science

Performance Analysis of SIMD vectorization of High-Order Finite-Element kernels

Gauthier Sornet, Sylvain Jubertie, Fabrice Dupros, Florent de Martin, Sébastien Limet

To cite this version:

Gauthier Sornet, Sylvain Jubertie, Fabrice Dupros, Florent de Martin, Sébastien Limet. Performance Analysis of SIMD vectorization of High-Order Finite-Element kernels. The 2018 International Conference on High Performance Computing & Simulation (HPCS 2018) - HPCS 2018, Jul 2018, Orléans, France. pp.423-430, 10.1109/HPCS.2018.00074 . hal-01915519v2

HAL Id: hal-01915519

https://hal.science/hal-01915519v2

Submitted on 9 Apr 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Performance Analysis of SIMD vectorization of High-Order Finite-Element kernels.

Gauthier Sornet(1,2), Sylvain Jubertie(1), Fabrice Dupros(2), Florent De Martin(2), Sebastien Limet(1)
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022.(1), BRGM, BP 6009, 45060 Orléans Cedex 2, France.(2)
Email: gauthier.sornet@univ-orleans.fr; sylvain.jubertie@univ-orleans.fr;f.dupros@brgm.fr;
f.demartin@brgm.fr; sebastien.limet@univ-orleans.fr

***Abstract*—Physics-based three-dimensional numerical simulations are becoming more predictive and are already essential for improving the understanding of natural phenomena, such as earthquakes, tsunami, flooding or climate change and global warming. Among the numerical methods available to support these simulations, Finite-Element formulations have been implemented in several major software packages. The efficiency of these algorithms remains a challenge due to the irregular memory access that prevents the squeezing out of the maximum level of performance out of current architectures. This is particularly true at the shared-memory level with several levels of parallelism and complex memory hierarchies. Despite significant efforts, automatic optimizations provided by compilers and high-level frameworks are often far from the performances obtained from hand-tuned implementations. In this paper, we have extracted a kernel from the EFISPEC software package developed at BRGM (the French Geological Survey). This application implements a high-order finite-element method to solve the elastodynamic equation. We characterize the performance of the extracted mini-app considering key parameters such as the order of the approximation, the memory access pattern or the vector length. Based on this study, we detail specific optimizations and we discuss the results measured as regards to the roofline performance model on Intel Broadwell and Skylake architectures.**

I. Introduction

The landscape of multicore processors or accelerators available to implement scientific applications leads to increasing concerns as regards to the real efficiency of key applications.

Each vendor is working on next-generation hardwares able to overcome exascale barriers. These upcoming architectures will probably combine high number of heterogeneous computing cores associated with advanced memory technologies. Efforts to prepare applications for this upcoming system should rely on a deep understanding of the algorithms to predict the performance. In fact, many works, like [1], [2], [3], [4], [5], deal with the efficiency concern from the interrelation between hardware and algorithms. Finite-element methods are representative of such situation, as these numerical approaches are at the heart of many open-source or commercial software packages [6], [7], [8]. One of the key features of this class of algorithm is the ability to handle complex geometrical object shapes. On top of the complex and unstructured meshes generally involved, the algorithm implements irregular data access to match with both the local and the global computational phases. Moreover, the order of the interpolation described by the basis functions significantly impacts the CPU-intensive element-by-element computational phase. Indeed, in [9], the authors show that increasing the polynomial approximation order improves the operational intensity of the kernel.

Our effort in this paper is to conduct a comprehensive study of the performance of High-Order Finite-Element numerical kernels. In case of an explicit time-marching algorithm, the summation of the element contributions (assembly phase) is a bottleneck. Our study target a representative Finite-Element Method (FEM) application. EFISPEC code ([10]) is developed at BRGM, the French Geological Survey, and solves the three-dimensional elastodynamic equations. The standard version of this code is implemented in Fortran2003 and heavily relies on the MPI library. The kernel extracted corresponds to the computation of the internal forces and represents a maximum of 90% of the total elapsed time. This paper extends previous works dealing with mono-core vectorization [11] and data-layout reorganization [12]. Our contributions in this paper are as follows:

- Theoretical performance for FEM numerical kernels based on the roofline performance model [13].
- Study of the impact of the order of approximation as regards to the operational intensity.
- Evaluation of both direct and indirect data access pattern on the peak performance.
- Comparison of the performance on Intel Broadwell and Skylake dual-socket computing nodes.

The paper proceeds as follows. Section II describes the related work. Section III details the elastodynamic equations and the spectral-element method implementation. Section IV details the challenges for efficient implementations on current architectures. In sections V and VI, we show the results obtained with AVX-2 and AVX-512 SIMD (Single Instruction Multiple Data) instructions. Finally, we conclude this study and present some future work.

II. Related Work

Though this work of evaluating the performance of a finite-element numerical kernel has already been done by using simpler approaches (for instance the Mantevo benchmark[1], optimizing applicative performance is a continuous effort as regards to emerging chips (growing vectorization capabilities for instance).

Besides the scientific problems, one major challenge is to face key breakthroughs on both the hardware and software sides that will drive the community to the Exaflops. For instance, the energy consumption wall, for systems built with several millions of heterogeneous cores, remains an open topic. Consequently, both the scalability and the efficiency (computation and memory movements) of current applications are critical.

As regards to earthquakes modeling on distributed-memory systems, several references ([14], [15], [16]) underline the scalability of explicit formulations to solve the elastodynamic equation. In this case, we benefit from limited amount of point-to-point communications between neighboring MPI sub domains. Significant works have been made to extend this parallel results on heterogeneous and low-power processor ([17], [18]), mainly for the simpler Finite-Difference method. For example, both auto-tuning and machine learning strategies ([19], [20]) have been considered to explore the usual and large space of optimization parameters (compiler flags, space and time tiling, memory mapping). These approaches have not yet been fully investigated for High-Order Finite Element methods. This is probably due to the complexity of the implementation of such kernels.

Nevertheless, few papers discuss low-level parallelism for Finite-Elements based methods ([21], [22], [23], [24], [25]). Among these contributions, mesh coloring algorithms have gained significant attention these last few years due to their ability to reduce synchronizations at the shared-memory level. The counterpart is the limited efficiency as regards to cache memory. In [12] we have extended this approach to parallel FEM assembly on multi-core architectures by implementing colored packs of elements along with advanced vectorization strategies.

Finally, one can change the data structure as detailed in [11]. In this reference, the authors compared the SOA (Structure of Arrays) data layout to the AOS (Array of Structures). This strategy shows limited impact on the performances that are mainly governed by the efficiency at the SIMD level. If we exploit knowledge from the physics as regards to specific geometries (for instance in geosciences), we can benefit from hybrid approaches that combines a structured mesh (with regular data access for the main part of the computation) and unstructured meshes (with irregular data access). This approach has been successfully implemented in [8].

[1] https://mantevo.org/MantevoOverview.pdf

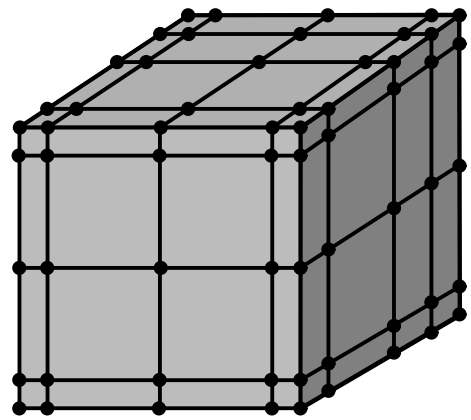

Figure 1: Referenced cube with $(4+1)^3 = 125$ GLL points.

III. EFISPEC: Spectral finite element solver

The spectral-element method (SEM) appeared more than 20 years ago in computational fluid mechanics [26], [27], [28]. The SEM is a specific formulation of the finite-element method for which the interpolated points and the quadrature points of an element share the same location. These points are the Gauss–Lobatto–Legendre (GLL) points, which are the $p+1$ roots of $(1-\xi^2)P_p'(\xi) = 0$, where P_p' denotes the derivative of the Legendre polynomial of degree p and ξ coordinate in the one-dimensional reference space $\Lambda = [-1, 1]$.

The generalization to higher dimensions is done through the tensorization of the one-dimensional reference space. In three dimensions, the reference space is the cube $\square = \Lambda \times \Lambda \times \Lambda$ (see Fig. 1).

The mapping from the reference cubes to a hexahedral element Ω_e is done by a regular diffeomorphism $\mathcal{F}_e : \square \to \Omega_e$. In a finite-element method, the domain of study is discretized by subdividing its volume Ω into welded non-overlapping hexahedral elements Ω_e, $e = 1, \ldots, n_e$ such that $\Omega = \cup_{e=1}^{n_e} \Omega_e$.

The elements Ω_e from the mesh of the domain. On the one hand, each element Ω_e has a local numbering of the GLL points ranging from 1 to $p+1$ along each dimension of the tensorization. On the other hand, the mesh has a unique global numbering ranging from 1 to N the global number of GLL points (see Fig. 2). The mapping from the local numbering to the global numbering is the so-called "assembly" phase of all finite-element calculations.

Each GLL point of an element Ω_e is redirected to a unique global number, $\forall \Omega_e$. When multiple elements share a common face, edge or corner, the assembly phase sums the local GLL value into the global numbering system.

In this article, the problem of interest is the equation of motion whose weak formulation is given by

$$\int_\Omega \rho \mathbf{w}^T \cdot \ddot{\mathbf{u}} \, d\Omega = \int_\Omega \boldsymbol{\nabla} \mathbf{w} : \boldsymbol{\tau} \, d\Omega - \int_\Omega \mathbf{w}^T \cdot \mathbf{f} \, d\Omega - \int_\Gamma \mathbf{w}^T \cdot \mathbf{T} \, d\Gamma$$

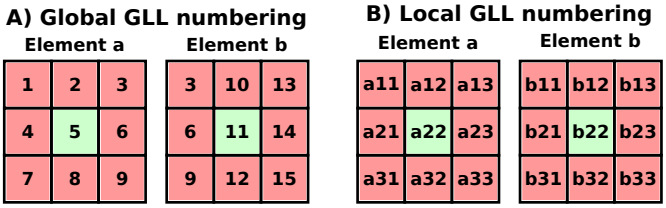

Figure 2: Global and local view of the GLL numbering.

where Ω and Γ are the volume and the surface area of the domain under study, respectively; ρ is the material density; $\mathbf{w}$ is the test vector; $\ddot{\mathbf{u}}$ is the second time-derivative of the displacement $\mathbf{u}$; $\boldsymbol{\tau}$ is the stress tensor; $\mathbf{f}$ is the body force vector and $\mathbf{T}$ is the traction vector acting on Γ. Superscript T denotes the transpose, and a colon denotes the contracted tensor product.

Our study focuses on the internal forces defined by (see [29])

$$\int_{\Omega_e} \boldsymbol{\nabla}\mathbf{w} : \boldsymbol{\tau}\, d\Omega_e \approx \sum_{\alpha=1}^{\mathrm{p+1}} \sum_{\beta=1}^{\mathrm{p+1}} \sum_{\gamma=1}^{\mathrm{p+1}} \sum_{i=1}^{3} \mathrm{w}_i^{\alpha\beta\gamma} \times \Bigg[\omega_\beta \omega_\gamma \sum_{\alpha'=1}^{\mathrm{p+1}} \left[\omega_{\alpha'} \mathcal{J}_e^{\alpha'\beta\gamma} \sum_{j=1}^{3} \left[\tau_{ij}^{\alpha'\beta\gamma} \partial_j \xi_{\alpha'} \right] \ell'_\alpha(\xi_{\alpha'}) \right] + \omega_\alpha \omega_\gamma \sum_{\beta'=1}^{\mathrm{p+1}} \left[\omega_{\beta'} \mathcal{J}_e^{\alpha\beta'\gamma} \sum_{j=1}^{3} \left[\tau_{ij}^{\alpha\beta'\gamma} \partial_j \eta_{\beta'} \right] \ell'_\beta(\eta_{\beta'}) \right] + \omega_\alpha \omega_\beta \sum_{\gamma'=1}^{\mathrm{p+1}} \left[\omega_{\gamma'} \mathcal{J}_e^{\alpha\beta\gamma'} \sum_{j=1}^{3} \left[\tau_{ij}^{\alpha\beta\gamma'} \partial_j \zeta_{\gamma'} \right] \ell'_\gamma(\zeta_{\gamma'}) \right] \Bigg]$$

with $\boldsymbol{\tau}$ the stress tensor ($= \boldsymbol{c} : \nabla\mathbf{u}$); $\mathcal{J}_e^{\alpha'\beta\gamma}$ the jacobian of an element Ω_e at the GLL points $\alpha'\beta\gamma$; ω_λ integration weight at the GLL point λ; ξ, η, ζ local coordinates along the three dimensions of the reference cube; ℓ'_λ derivative of the Lagrange polynomial at the GLL point λ. $\boldsymbol{c}$ is the elastic tensor and $\nabla\mathbf{u}$ is the gradient of the displacement defined by

$$\partial_i \mathrm{u}_j\left(\xi_\alpha \eta_\beta \zeta_\gamma\right) = \left[\sum_{\sigma=1}^{\mathrm{p+1}} u_j^{\sigma\beta\gamma} \ell'_\sigma\left(\xi_\alpha\right) \partial_i \xi_{\alpha\beta\gamma} \right] + \left[\sum_{\sigma=1}^{\mathrm{p+1}} u_j^{\alpha\sigma\gamma} \ell'_\sigma\left(\eta_\alpha\right) \partial_i \eta_{\alpha\beta\gamma} \right] + \left[\sum_{\sigma=1}^{\mathrm{p+1}} u_j^{\alpha\beta\sigma} \ell'_\sigma\left(\zeta_\alpha\right) \partial_i \zeta_{\alpha\beta\gamma} \right]$$

IV. Challenges

A. Data access

Our numerical simulations are computed over a domain. The domain is discretized in GLL points following a 3D mesh. These GLLs contains the physical values. Each element represents a subspace of the simulated domain.

```
o = Polynomial Order;
Direct_GLL(i,j,k,e)=
 globalGLL[(((e*o+k)*o+j)*o+i)];

p = Polynomial Order+1;
Indirect_GLL(i,j,k,e)=
 globalGLL[ElmIdx[e][((k*p+j)*p+i)]];
```

Figure 3: The direct and indirect GLL access formulation.

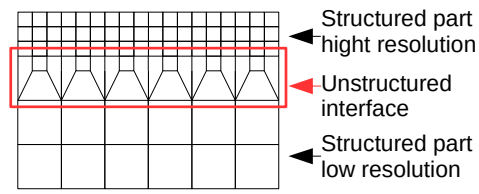

Figure 4: Two-dimensional mesh interface exemple between two levels of resolution.

We assembled each element following its GLL and parameters. Basically, each element has a 3D grid of indices that refer to GLL points from a global GLL list. The elements from structured meshes can be allocated following a three-dimensional regular grid as in Fig 5. Thus, the GLL address is found with the direct address formulation Fig. 5. Therefore, without any GLL index, the operational intensity is higher. Unstructured meshes do not allow to allocate the GLL data as a regular 3D grid. Thus, the GLL data have to be assessed by the indirect address formulation Fig. 3. We only evaluate the indirection over costs. Therefore, our input GLL data are always the same regular grid as in the figure 5. The difference between direct and indirect implementations comes from the GLL data access formulation Fig. 3. The EFISPEC application has to deal with unstructured meshes. His original implementation requires to use an indirect GLL access.

The Geoscience meshes can have some large computing zones with very local regions of interest. Thus, some unstructured sub-meshes can be located at the interfaces between two different regions represented by structures meshes a shown by the figure 4. Therefore, it is interesting to evaluate the performances of a regular access GLL structure as shown by the figure 5. To summarize, the main idea is to separate the computation between structured and unstructured areas of the same mesh. Although, the unstructured mesh areas cannot be allocated as a regular 3D grid, it is still possible to optimize its GLL allocation orders.In fact, the reverse Cuthill–McKee algorithm is used to optimize the neighboring locality [30]. We also assumed that METIS optimizes the element locality.

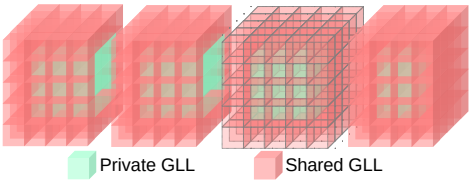

Figure 5: Gauss–Lobatto–Legendre (GLL) direct access structure

B. Vectorization

The vectorization of our kernel is mainly described in [12], [11]. It is based on the use of intrinsics since we observed that compilers are not able to vectorize the code efficiently. The principle is to compute several elements at the same time depending on the SIMD unit width. In our previous work, we have implemented this approach on AVX and AVX-512 capable architectures. In this paper, we also consider the AVX-512/256 instruction subset which is part of the AVX-512 instruction set but process only 256 bits of data at a time instead of 512. The AVX-512/256 subset extends the AVX2 instruction set by adding scatter intrinsics to implement the indirect data access in a simpler way. It allows to gather and scatter GLL values between global and local arrays as detailed from in sub-section IV-A. Thus, we can study the impact of doubling the SIMD width with the same instructions.

C. Operational intensity

We compare our different kernel implementations using their Operational Intensities (O.I.), as defined in [13]. This O.I. is defined as the ratio between the number of floating point operations and the number of bytes loaded from the DRAM.
Each element of order o is composed of $(o+1)^3$ GLLs. Computing an element consists in loading its GLL values and applying floating point operations. Data loaded from DRAM may be reused by the operations according to the element order as detailed in the table I.
Computing a three dimensional element of order 4, as the one in figure 1, requires $(4+1)^3 = 125$ GLL values to load from DRAM and some additional parameters (GLL weights and Lagrange derivatives). It represents 11120 bytes of data on which 48150 floating point operations are applied. Thus, the operational intensity of an order 4 element is $48150/11120 = 4.33$. We determine this O.I. for the different kernel implementations and compare them with each other.

V. Experimental setup

A. Experimental context

Two different platforms are considered. The first one is equipped with two Intel Xeon Gold 6148 processors based on the Skylake architecture. The second one comes with two Intel Xeon 2697v4 processors based on the previous Broadwell architecture. Table II contains the characteristics of both platforms. Hyperthreading is disabled on both platforms.

Order	Flop/Element	Data access	Byte/Element	O.I
2	7974	Regular	2316	3.44
		Irregular	2424	3.29
4	48150	Regular	10620	4.53
		Irregular	11120	4.33
8	411966	Regular	61596	6.69
		Irregular	64512	6.39

Table I: Operational Intensity (O.I) with respect to the order of the elements. The irregular version requires an additional indirection array whereas the regular one uses a direct access to the GLLs.

Platform	**Skylake**	**Broadwell**
physical cores	2×20	2×18
base frequency	2.4GHz	2.3GHz
AVX frequency	1.9GHz	2.0GHz
AVX-512 frequency	1.6GHz	-

Table II: Platform characteristics

To compare the O.I. of each of our implementations, we propose to use the Original Roofline Model (ORM) to study the performance and the limitations of our implementations. However, in [5], [3], [4], authors show some limitations of the ORM when the model is used to drive the optimization process. These works propose other roofline models, such as the Cache Aware Roofline Model (CARM) and Locality Aware Roofline Model (LARM), which take into account more architectural details. In our case, we only use the ORM to discuss the relative performance of our implementations and not to drive the optimization process.

The rooflines has been established for both platforms from a BLAS SGEMM benchmark and the STREAM benchmark [31]. Figure 6 shows the resulting rooflines. The Broadwell and Skylake platforms achieve a peak performance of respectively 2,314 GFLOPS and 3,826 GFLOPS .

We consider two different compilers in our study: ICC 2017 and Clang 5. Since results obtained with both compilers are not significantly different and exhibit a similar behavior, we only present results obtained with Clang 5.

B. Kernels

We have derived several versions from the original EFISPEC kernel to study the impact of the approximation order and mesh structure. Two data access patterns are considered:

1) Irregular: the original EFISPEC access pattern based on an indirection array to access GLL elements in an unstructured mesh, also used in common FEM implementations.
2) Regular: GLL elements are stored in a regular pattern suitable to represent a structured mesh.

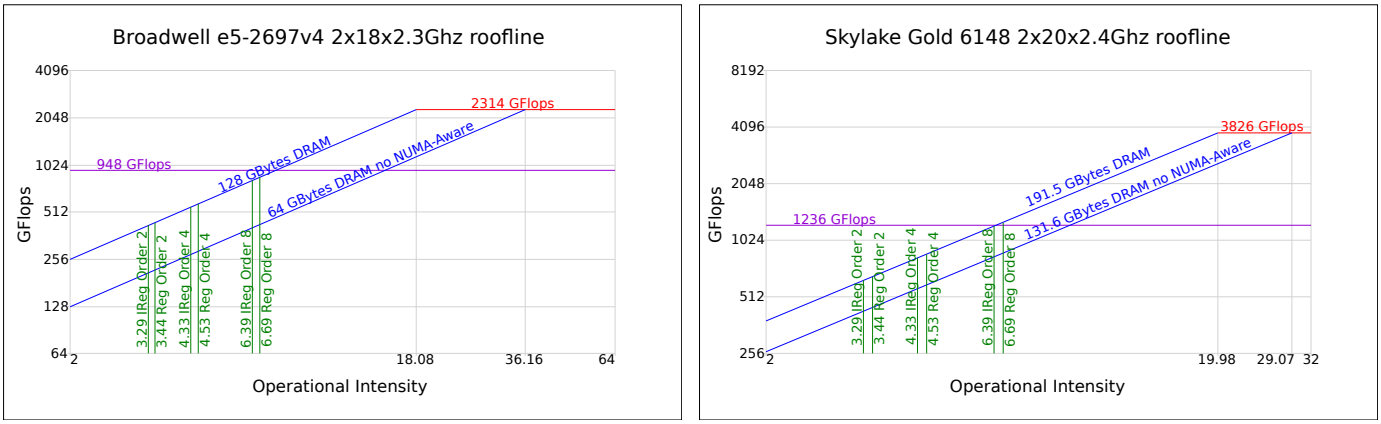

Figure 6: Rooflines for the Broadwell and Skylake platforms.

For each access pattern, we derive three variants for orders 2, 4 and 8. Each variant comes in three SIMD versions:

1) **AVX2**: it operates on 256-bit registers and is able to run on both the Broadwell and Skylake platforms,
2) **AVX-512**: it operates on 512-bit registers thus it is only able to run on the Skylake platform,
3) **AVX-512/256**: it is the same as the AVX2 version but uses a specific 256-bit SIMD scatter instruction (accessible through the `__mm256_i32scatter_ps` intrinsics) only available in the AVX-512 instruction set.

Comparing the AVX-512 and AVX-512/256 versions allows to study the gain brought by doubling the width of the SIMD unit. Note that, as reported in table II, cores adapt their frequency depending on the kind of SIMD instructions processed. For each kernel, the corresponding operational intensity, as indicated in table I, is reported into the rooflines. On both platforms, rooflines show that our kernels are memory-bound. Additionally, we have introduced a peak application performance for each architecture. These values are computed with a small example able to fit into the cache. In this case we measure a peak value of 948 GFLOPS on Broadwell and 1,236 GFLOPS on Skylake. Based on these local peak performances, we can observe that the implementation of the order 8 versions are CPU-limited in almost all cases.

VI. Experimental results

A. Impact of the data access pattern

In this section, we comment on the impact of direct or indirect data access pattern on the performance observed. One of the key point is coming from the fact that an indirect data access implementation involves more data movement compared to a direct version (see Table I). In this case, each element requires to be transformed to a local representation before the compute and assembly phase. Even if the vectorization can deal with indirect loads and stores, we still need to retrieve the indices from DRAM to cache memory.

From figure 7, that represents the peak performance with respect to the data access strategy on Intel Broadwell and Skylake hardware, we can extract two main outcomes. First of all, the best results as regards to both scaling and peak performance are obtained with the direct and simpler data access pattern. This is expected as operational intensities of these algorithms are higher (for instance the O.I for the order 2 version is 3.44 in the direct case and 3.29 for the indirect case). Moreover these implementations can fully benefit from vectorization thanks to the regularity in the memory accesses.

Second of all, regardless of the polynomial order, the gap between direct and indirect data access versions is higher on Skylake. The direct data access pattern version outperforms the indirect one by 47% on Skylake but only by 34% on Broadwell for the higher polynomial order. Similar trend could be observed for lower polynomial orders as the benefit from regular data access pattern is clearly growing with the size of the SIMD vector.

B. Impact of the polynomial order

The main objective of this section is to discuss the impact of the order of approximation for the element-by-element computation. One common suggestion is that the extra computation coming with a higher polynomial order that improves the O.I factor.If we study the shape of the plots from Broadwell and Skylake platforms (see Figure 7), we can observe a saturation of the peak performance as we increase the number of cores. This behavior could be explained by the roofline model and underlines that the higher order versions are more likely to be CPU-bound for direct access pattern. For irregular versions, the results on the Skylake platform are different as the lower order version performs better than the others. A similar behavior is observed on the Broadwell platform also with almost the same level of performance between both implementations.

These results do not match the operational intensity values we have determined (3.29 for order 2 and 6.39 at

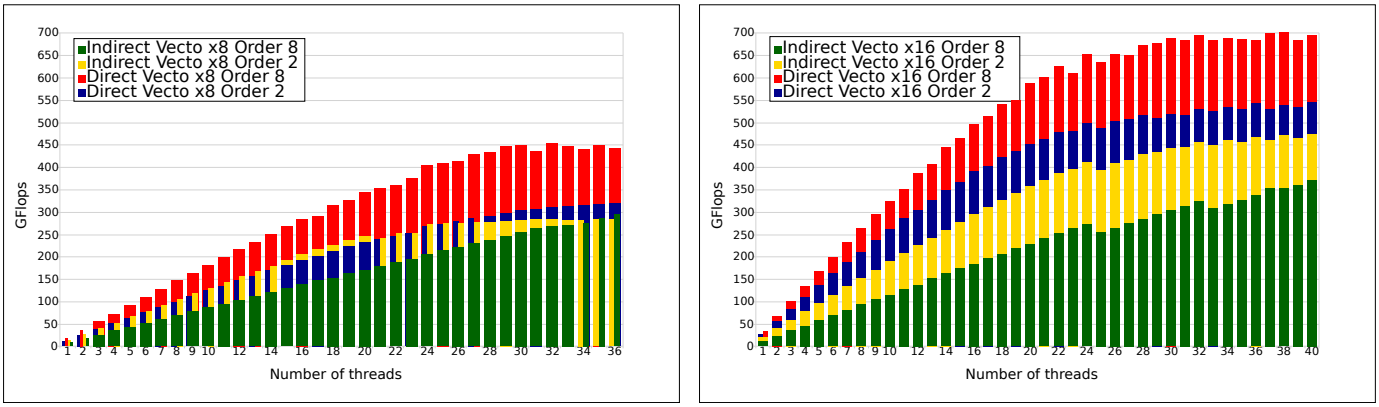

Figure 7: Comparison of the implementations with direct or indirect data access pattern on Broadwell (left) and Skylake (right) platforms.

order 8). Indeed, profiling the code with VTune shows that the time spent in cache accesses is multiplied by two with the indirect data access from order 2 to 8.When we increase the order of approximation, the amount of data is significantly larger, exceeding cache size and generating much more memory traffic with slower levels of memory. To illustrate this, Fig. 8 compares local or remote memory access situations on our dual-socket shared-memory systems. If we compare both memory mapping strategies, the NUMA penalty leads to a reverse ranking of the different implementations as regards to their peak performances.

C. Impact of the vectorization

The support of the AVX512 allows the hardware to compute 16 single precision floats by instruction. We recall from [12] that automatic optimizations provided by the compilers hardly reach 140 GFLOPS on both platforms, this represents less than 5% of the theoretical peak performance. Figure 9 compares the performance between 256-bit and 512-bit SIMD instructions.
The AVX-512 with irregular data access version exhibits a speedup of 1.20 for order 2 and 1.13 for order 8 compared to AVX-512/256 implementations. As expected, the performance gain from the use of AVX-512 instructions is lower than 2 as regards to AVX-512/256. Conversely, the direct data access implementation benefits more from this improvement as we measure almost a speedup of 2.

D. Comparison with roofline model results

The operational intensities of our different implementations are shown in Figure 6. Most of the kernels are bounded by the memory bandwidth. Therefore, we estimate the maximal GFLOPS peak performance of an implementation on a given platform by multiplying the peak memory bandwidth by the O.I factor. For instance, the implementation based on an indirect data access pattern and an order 2 approximation (IReg Order 2) shows a peak performance of $3.29 \times 128 = 421.12\,GFLOPS$.
The outcomes of the previous sections underline the following points. Firstly, the performance levels expected from the operational intensity values are far from being reproduced during real experiments. For instance, the ratio between the O.I for order 2 and order 8 implementations is 1.94. For simpler data access pattern and one core, measured results approximately match with the O.I ratios (i.e.1.51 on Broadwell and 1.21 on Skylake). When more computing cores are involved, the quality of the theoretical model tends to diminish due to a lack of refinement as regards to complex interactions at the shared-memory level.
Secondly, for the indirect memory access pattern, the ratio is reversed in almost all cases. That underlines the limitation of our model to take into account complex data movements on hierarchical architectures.

Data	AVX2	Order	1 thread	36 threads
Indirect access	/256	2	12.9 GFLOPS	284.1 GFLOPS
		8	8.6 GFLOPS	295.0 GFLOPS
Direct access	/256	2	11.9 GFLOPS	319.4 GFLOPS
		8	18.0 GFLOPS	442.3 GFLOPS

Table III: SIMD 256 bits performances from the Broadwell platform.

Data	AVX512	Order	1 thread	40 threads
Indirect access	/256	2	16.4 GFLOPS	474.0 GFLOPS
		8	10.4 GFLOPS	393.9 GFLOPS
	/512	2	19.6 GFLOPS	474.0 GFLOPS
		8	11.7 GFLOPS	371.2 GFLOPS
Direct access	/256	2	14.3 GFLOPS	458.4 GFLOPS
		8	19.1 GFLOPS	610.4 GFLOPS
	/512	2	28.2 GFLOPS	546.0 GFLOPS
		8	34.0 GFLOPS	696.0 GFLOPS

Table IV: SIMD 256 bits and SIMD 512 bits performances from the Skylake platform.

VII. Conclusion and future work

Optimizing the most popular numerical methods for parallel architectures is a continuous effort. Though the

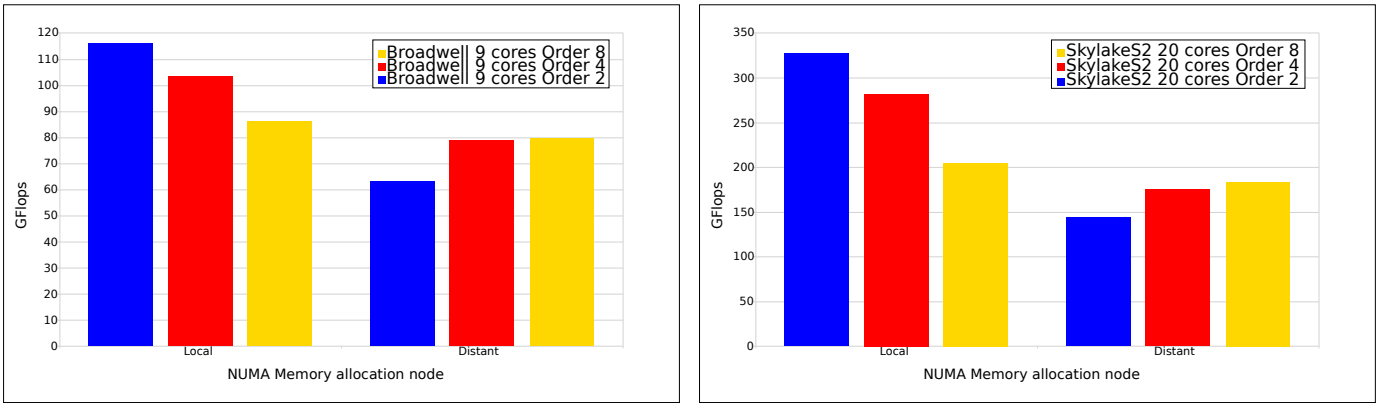

Figure 8: Comparison of local or remote memory access strategies with the indirect data access pattern on Broadwell and Skylake platforms.

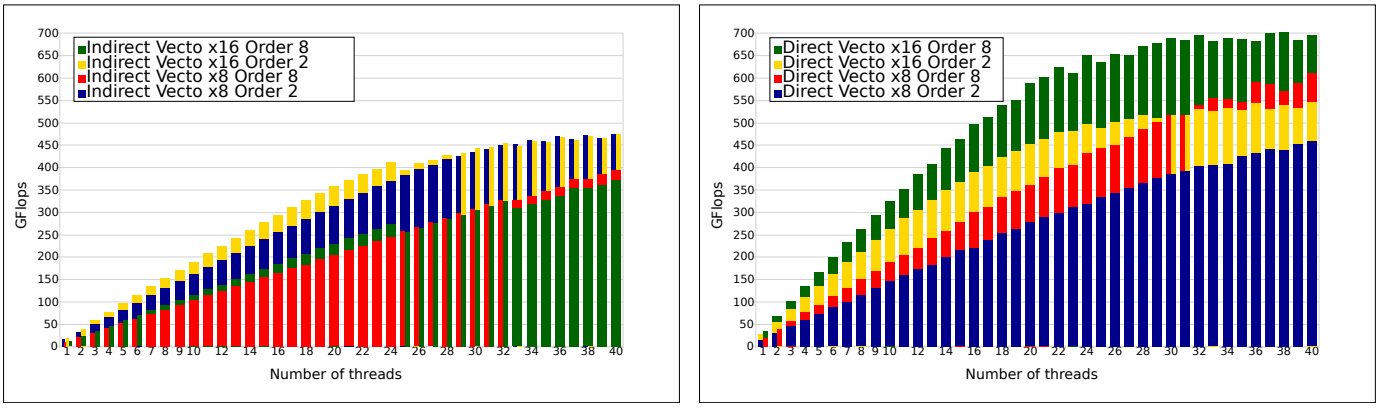

Figure 9: Comparison between AVX512 and AVX-512/256 with regular and irregular data access on the Skylake platform.

main characteristics of these kernels are already well known, finding the best algorithms or implementations on each architecture can be viewed as a quest. Our paper underlines some of the key aspects of the Finite-Element Method by exploiting the popular roofline theoretical performance model. As explained in the sub-section V-A, we are comparing our results with the ORM but further investigations should be done with the CARM and the LARM to clearly identify the involved bottlenecks. We also study the correlation between the operational intensity and the real performance on two leading dual-socket Intel platforms.

Based on the theoretical model, we have underlined the impact of the data access patterns that represent a major bottleneck as regards to the expected performance. Additionally, from real experiments, we have demonstrated the limited improvements coming from higher polynomial orders on current architectures. One major outcome of this study is also the tremendous complexity of performance prediction, particularly on available chips that combine several levels of parallelism and cache hierarchies.

Obviously, the proposed optimizations would benefit from an integration in a high-level framework in order to ease their integration in a large number of scientific applications. We also believe that significant efforts should be made at the performance modeling level, probably with a wider usage of low-level emulation tools, to bring key applications up to speed on emerging processors.

Acknowledgment

The authors thank Philippe Thierry, principal engineer at Intel, for many interesting discussions and for providing us access to Broadwell and Skylake platforms. We would also like to thank Faiza Boulahya from BRGM (French Geological Survey) for relevant comments on this paper. The work of Gauthier Sornet is co-funded by the Region Centre-Val de Loire and BRGM.

References

[1] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick, "Optimization of a lattice boltzmann computation on state-of-the-art multicore platforms," *Journal of Parallel and Distributed Computing*, vol. 69, no. 9, pp. 762 – 777, 2009. Best Paper Awards and Panel Summary: 22nd International Parallel and Distributed Processing Symposium (IPDPS 2008).

[2] K.-H. Kim, K. Kim, and Q.-H. Park, "Performance analysis and optimization of three-dimensional fdtd on gpu using roofline model," *Computer Physics Communications*, vol. 182, no. 6, pp. 1201 – 1207, 2011.

[3] D. Marques, H. Duarte, A. Ilic, L. Sousa, R. Belenov, P. Thierry, and Z. A. Matveev, "Performance analysis with cache-aware roofline model in intel advisor," in *2017 International Conference on High Performance Computing Simulation (HPCS)*, pp. 898–907, July 2017.

[4] N. Denoyelle, B. Goglin, A. Ilic, E. Jeannot, and L. Sousa, "Modeling Large Compute Nodes with Heterogeneous Memories with Cache-Aware Roofline Model," in *High Performance Computing systems - Performance Modeling, Benchmarking, and Simulation - 8th International Workshop, PMBS 2017*, vol. 10724 of *Lecture Notes in Computer Science*, (Denver (CO), United States), pp. 91–113, Springer, Nov. 2017.

[5] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware roofline model: Upgrading the loft," *IEEE Computer Architecture Letters*, vol. 13, pp. 21–24, Jan 2014.

[6] N. Möller, E. Petit, L. Thébault, and Q. Dinh, "A case study on using a proto-application as a proxy for code modernization," *Procedia Computer Science*, vol. 51, pp. 1433 – 1442, 2015. International Conference On Computational Science, ICCS 2015.

[7] D. Komatitsch, D. Michéa, and G. Erlebacher, "Porting a high-order finite-element earthquake modeling application to {NVIDIA} graphics cards using {CUDA}," *Journal of Parallel and Distributed Computing*, vol. 69, no. 5, pp. 451 – 460, 2009.

[8] T. Ichimura, K. Fujita, P. E. B. Quinay, L. Maddegedara, M. Hori, S. Tanaka, Y. Shizawa, H. Kobayashi, and K. Minami, "Implicit nonlinear wave simulation with 1.08t dof and 0.270t unstructured finite elements to enhance comprehensive earthquake simulation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, (New York, NY, USA), pp. 4:1–4:12, ACM, 2015.

[9] F. Krużel and K. Banaś, "Vectorized opencl implementation of numerical integration for higher order finite elements," *Computers & Mathematics with Applications*, vol. 66, no. 10, pp. 2030 – 2044, 2013. ICNC-FSKD 2012.

[10] F. De Martin, "Verification of a spectral-element method code for the southern california earthquake center loh.3 viscoelastic case," *Bull. Seism. Soc. Am.*, vol. 101, no. 6, pp. 2855–2865, 2011.

[11] S. Jubertie, F. Dupros, and F. D. Martin, "Vectorization of a spectral finite-element numerical kernel," in *Proceedings of the 4th Workshop on Programming Models for SIMD/Vector Processing, WPMVP@PPoPP 2018, Vienna, Austria, February 24, 2018*, pp. 8:1–8:7, 2018.

[12] G. Sornet, S. Jubertie, F. Dupros, F. De Martin, P. Thierry, and S. Limet, "Data-layout reorganization for an efficient intra-node assembly of a Spectral Finite-Element Method," in *PDP2018*, (Cambridge UK, United Kingdom), Mar. 2018.

[13] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, pp. 65–76, Apr. 2008.

[14] D. Roten, Y. Cui, K. B. Olsen, S. M. Day, K. Withers, W. H. Savran, P. Wang, and D. Mu, "High-frequency nonlinear earthquake simulations on petascale heterogeneous supercomputers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016*, pp. 957–968, 2016.

[15] S. Tsuboi, K. Ando, T. Miyoshi, D. Peter, D. Komatitsch, and J. Tromp, "A 1.8 trillion degrees-of-freedom, 1.24 petaflops global seismic wave simulation on the K computer," *IJHPCA*, vol. 30, no. 4, pp. 411–422, 2016.

[16] J. Tobin, A. Breuer, A. Heinecke, C. Yount, and Y. Cui, "Accelerating seismic simulations using the intel xeon phi knights landing processor," in *High Performance Computing - 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18-22, 2017, Proceedings*, pp. 139–157, 2017.

[17] D. Göddeke, D. Komatitsch, M. Geveler, D. Ribbrock, N. Rajovic, N. Puzovic, and A. Ramírez, "Energy efficiency vs. performance of the numerical solution of pdes: An application study on a low-power arm-based cluster," *J. Comput. Physics*, vol. 237, pp. 132–150, 2013.

[18] M. Castro, E. Francesquini, F. Dupros, H. Aochi, P. O. A. Navaux, and J. Méhaut, "Seismic wave propagation simulations on low-power and performance-centric manycores," *Parallel Computing*, vol. 54, pp. 108–120, 2016.

[19] P. Souza, L. Borges, C. Andreolli, and P. Thierry, "Chapter 24 - portable explicit vectorization intrinsics," in *High Performance Parallelism Pearls* (J. Reinders, , and J. Jeffers, eds.), pp. 463 – 485, Boston: Morgan Kaufmann, 2015.

[20] F. Martínez-Martínez, M. J. Rupérez-Moreno, M. Martínez-Sober, J. A. S. Llorens, D. Lorente, A. J. Serrano, S. Martínez-Sanchis, C. M. Aranda, and J. D. Martín-Guerrero, "A finite element-based machine learning approach for modeling the mechanical behavior of the breast tissues under compression in real-time," *Comp. in Bio. and Med.*, vol. 90, pp. 116–124, 2017.

[21] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder, "Sparse matrix solvers on the gpu: Conjugate gradients and multigrid," *ACM Trans. Graph.*, vol. 22, pp. 917–924, July 2003.

[22] C. Farhat and L. Crivelli, "A general approach to nonlinear fe computations on shared-memory multiprocessors," *Computer Methods in Applied Mechanics and Engineering*, vol. 72, no. 2, pp. 153 – 171, 1989.

[23] L. Thébault, E. Petit, M. Tchiboukdjian, Q. Dinh, and W. Jalby, "Divide and conquer parallelization of finite element method assembly," in *Parallel Computing: Accelerating Computational Science and Engineering (CSE), Proceedings of the International Conference on Parallel Computing, ParCo 2013, 10-13 September 2013, Garching (near Munich), Germany*, pp. 753–762, 2013.

[24] C. Cecka, A. J. Lew, and E. Darve, "Assembly of finite element methods on graphics processors," *International Journal for Numerical Methods in Engineering*, vol. 85, no. 5, pp. 640–669, 2011.

[25] G. R. Markall, A. Slemmer, D. A. Ham, P. H. J. Kelly, C. D. Cantwell, and S. J. Sherwin, "Finite element assembly strategies on multi-core and many-core architectures," *International Journal for Numerical Methods in Fluids*, vol. 71, no. 1, pp. 80–97.

[26] A. T. Patera, "A spectral element method for fluid dynamics: laminar flow in a channel expansion," *J. Comput. Phys.*, vol. 54, pp. 468–488, 1984.

[27] Y. Maday and A. T. Patera, "Spectral element methods for the incompressible navier-stokes equations," *State of the art survey in computational mechanics*, pp. 71–143, 1989.

[28] P. F. Fischer and E. M. Rønquist, "Spectral-element methods for large scale parallel Navier-Stokes calculations," *Comput. Methods Appl. Mech. Engrg.*, vol. 116, pp. 69–76, 1994.

[29] D. Komatitsch and J. Tromp, "Spectral-element simulations of global seismic wave propagation-I. Validation," *Geophys. J. Int.*, vol. 149, no. 2, pp. 390–412, 2002.

[30] D. Komatitsch, J. Labarta, and D. Michéa, *A Simulation of Seismic Wave Propagation at High Resolution in the Inner Core of the Earth on 2166 Processors of MareNostrum*, pp. 364–377. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[31] J. McCalpin, "Sustainable memory bandwidth in high-performance computers," 1995.