Performance optimization of fully anisotropic elastic wave propagation on 2nd Generation Intel® Xeon Phi™ processors

Albert Farres*† Claudia Rosas* Mauricio Hanzich*
*Barcelona Supercomputing Center (BSC)
†Universitat Politècnica de Catalunya (UPC)
Spain
{albert.farres,claudia.rosas,mauricio.hanzich}@bsc.es

Alejandro Duran
Intel Corporation Iberia
Spain
alejandro.duran@intel.com

Charles Yount
Intel Corporation
USA
chuck.yount@intel.com

Keywords*—*Stencil-based wave propagation, Performance optimizations, Intel Xeon Phi, Fully Staggered Grid

I. Extended Abstract

In the last few years, acoustic isotropic wave propagation has been the preferred simulation engine for 3D full-wave field modelling-based applications. The reason for its success over better approximations is the lower computational cost it entails. However, recent trends in seismic imaging rely on an improved physical model that represents the Earth no more as a rigid body but as an elastic and anisotropic one. Also, moving the wave propagation simulation closer to the real physics of the problem results in a significant increment of the needed computational resources. New hardware alternatives appear as a potential solution to satisfy the high demands of the computing power of the elastic, anisotropic, wave propagation engine. Also, the last decade has seen a trend on building systems with dedicated devices and accelerators, which produce a good FLOPs/Watt ratio. One of the most promising HPC alternatives comes from Intel® Phi™ product family.

This work shows several optimization strategies evaluated and applied to an elastic wave propagation engine, based on a Fully Staggered Grid, running on the latest Intel Xeon Phi processors, the second generation of the product (code-named Knights Landing). The developed propagator is able to reproduce elastic wave propagation, even for an arbitrary anisotropy.

A. Elastic FSG wave propagator algorithm

The Algorithm 1 describes our implementation of the FSG wave propagator using finite differences. It has been implemented from scratch inside YASK [1] framework using its domain-specific language (DSL), and it is now available as one of the example solutions included in the open-source package.[1]

[1]https://github.com/intel/yask

Algorithm 1 FSG wave propagator using finite differences

```
1:  function UPDATE_VELOCITIES
2:      for all cells do
3:          12 × density interpolations
4:          12 × derivative calcs. based on stresses
5:          12 × velocity updates
6:      end for
7:  end function
8:  function UPDATE_STRESSES
9:      for all cells do
10:         84 × coefficient interpolation
11:         28 × derivative calcs. based on velocities
12:         24 × stress updates
13:     end for
14: end function
15: for all timesteps do
16:     update_velocities ()
17:     update_stresses ()
18: end for
```

B. The Intel Xeon Phi processor

For this study we are using a system with a CPU from the Intel Xeon Phi x200 processor family (code-named Knights Landing). These processors have a multi-core architecture with up to 36 tiles per package connected through a 2D mesh between them and the memory controllers. Each tile is composed of two cores that share a 512 MB L2 cache and an agent that connects the tile to the mesh. Each core appears as four logical CPUs through hyper-threading. All hyper-threads on a core share a first-level (L1) cache. The cores are capable of issuing two instructions per cycle out-of-order, including vector and memory instructions. The Intel Xeon Phi architecture implements the Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instruction set. To be able to provide the cores with enough data to feed the computing capabilities, the Intel Xeon Phi processors may be configured with an integrated on-package Multi-Channel DRAM (MCDRAM) memory of up to 16 GiB, which can deliver up to 490 GB/s of bandwidth. The main DDR4 memory on the same platform can deliver about 90 GB/s.

C. Evaluation Methodology

The set of optimizations applied to the code combine classic and widely used performance improvement approaches, such as: blocking, prefetching and scheduling, with

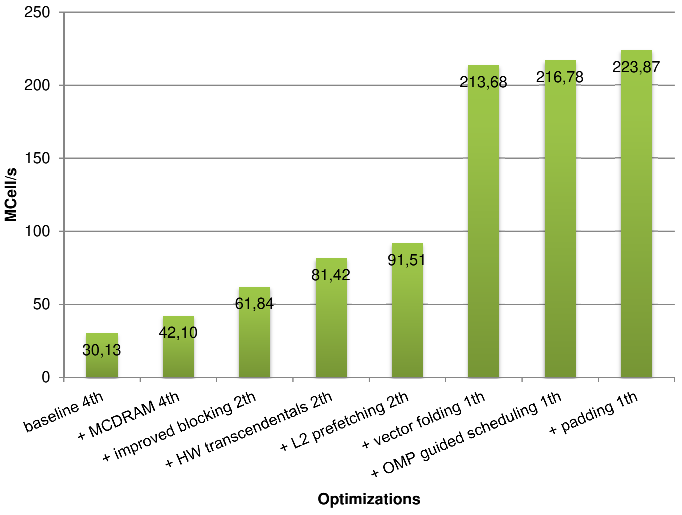


Fig. 1. FSG wave propagator throughput on Intel Xeon Phi

architecture-dependent optimizations like approximate reciprocal, and mechanisms from the state-of-the-art like vector folding. Additional details of the optimization methods are described below.

Baseline. The reference performance baseline was obtained by executing a fairly optimized version of FSG with parallelization using hyper-threading and blocking. This version uses the default compilation and run-time configuration provided by YASK. **MCDRAM memory.** Bandwidth-bound applications can easily benefit from the MCDRAM of the Intel Xeon Phi processors, exhibiting better execution times than with conventional DDR DRAM. **Improved blocking.** To improve temporal data reuse and reduce the memory bandwidth requirements per updated grid point, we implemented a loop blocking strategy transforming the memory domain of our problem into smaller chunks, rather than sequentially traversing through the entire memory domain. **Threads per core.** On Intel Xeon Phi processors is possible to place up to 4 threads per core. Reducing the numbers of threads per core to just 1 gave us the best performance. **Approximate reciprocal.** We approximated divisions using the reciprocal intrinsic instructions available in Intel AVX-512-ER which provide a faster implementation restraining the error. **Prefetching.** While the Intel Xeon Phi processor automatically prefetches data into the L1 and L2 caches it is possible to improve the memory behavior by using software prefetching. We have enabled aggressive software prefetching for L1 data cache. **Vector folding**. This method stores a small multi-dimensional block of data in each vector size memory region compared to the single dimension in the traditional approach. Therefore memory accesses required are reduced by increasing overlap between points for the stencil computation [1]. **OpenMP scheduling.** In most implementations the default loop scheduling algorithm in OpenMP is static, but it is not necessarily the best one. We evaluated the three most relevant loop scheduling strategies: static, dynamic and guided. The best performance in our case resulted from using dynamic scheduling. **Padding.** Padding improves performance by carefully aligning data in memory at the expense of using extra memory.

D. Results

Optimizations were enabled incrementally in the given order. Major changes, such as total number of threads used per core, or a different, sometimes additional optimization are clearly stated. To avoid potential outliers, all the experiments were executed in an exclusive node and repeated several times. The algorithm and the machine were found to be stable, showing only negligible variations between experiments. The throughput results of FSG are summarized in Fig. 1. It represents the maximum throughput obtained for each optimization and for the baseline implementation using DDR memory. The number of threads used in each case is shown after each configuration (e.g., “4th” indicates four threads per core). Baseline and MCDRAM ran with 4 threads per core; improved blocking, hardware trascendentals and L2 prefetching used 2 threads per core; and the remaining experiments ran with 1 thread per core.

E. Conclusion

We have shown a set of optimizations, applied to a Finite Difference Numerical method solving elastic wave propagation equations on the second generation Intel Xeon Phi processor. Moreover, the proposed scheme for solving the elastic equation supports arbitrary anisotropy at a higher computational cost when compared to more traditional approaches to address this problem. The evaluated set of optimizations ranges from memory to compute optimizations. Our findings indicated that, compared to a conventionally-optimized version, we were able to achieve 7× more FLOPS and 8× more bandwidth with all implemented optimizations. In consequence, we reached 75% of the maximum attainable FP performance at our current operational intensity.

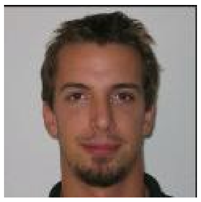
II. Acknowledgment

This work has been published in proceedings of the International Parallel and Distributed Processing Symposium Workshops (IPDPS), 2018 [2].

References

[1] C. Yount, “Vector folding: Improving stencil performance via multi-dimensional simd-vector representation,” in *IEEE 17th International Conference on High Performance Computing and Communications (HPCC)*, Aug 2015, pp. 865–870.

[2] A. Farres, C. Rosas, M. Hanzich, A. Duran, and C. Yount, “Performance optimization of fully anisotropic elastic wave propagation on 2nd Generation Intel Xeon Phi processors,” in *Proceedings of the 32nd IEEE International Parallel and Distributed Processing Symposium Workshops*, 2018, pp. 1033–1042.



Albert Farres is an engineer at Barcelona Supercomputing Center, the Spanish National Supercomputing Institute. He is currently researching and developing seismic imaging tools for the oil & gas industry. He has a MSc degree and a Bachelor in Computer Science from the Universitat Politècnica de Catalunya.