High Order Seismic Simulations on the Intel Xeon Phi Processor (Knights Landing)

Alexander Heinecke[1], Alexander Breuer[2], Michael Bader[3], and Pradeep Dubey[1]

[1] Intel Corporation, 2200 Mission College Blvd., Santa Clara 95054, CA, USA
[2] University of California, San Diego, 9500 Gilman Dr., La Jolla 92093, CA, USA
[3] Technische Universität München, Boltzmannstr. 3, D-85748 Garching, Germany

Abstract. We present a holistic optimization of the ADER-DG finite element software SeisSol targeting the Intel® Xeon Phi™ x200 processor, codenamed Knights Landing (KNL). SeisSol is a multi-physics software package performing earthquake simulations by coupling seismic wave propagation and the rupture process. The code was shown to scale beyond 1.5 million cores and achieved petascale performance when using local time stepping for the computationally heavy seismic wave propagation. Advancing further along these lines, we discuss the utilization of KNL's core features, the exploitation of its two-level memory subsystem (which allows for efficient out-of-core implementations), and optimizations targeting at KNL's 2D mesh on-die interconnect. Our performance comparisons demonstrate that KNL is able to outperform its previous generation, the Intel® Xeon Phi™ coprocessor x100 family, by more than 2.9 × in time-to-solution. Additionally, our results show a 3.4 × speedup compared to latest Intel® Xeon® E5v3 CPUs.

Keywords: high-order, vectorization, ADER, discontinuous Galerkin, finite element method, Intel Xeon Phi, Knights Landing, KNL

1 Introduction

The understanding of earthquake dynamics is greatly supported by highly resolved, coupled simulations of the rupture process and seismic wave propagation. Requirements in resolution are pushed by detailed discretizations of complex geometric features, accurate representations of material heterogeneities and the need for resolved, high frequencies. This grand challenge of seismic modeling requires a large amount of computational resources. Optimal utilization by software is imperative.

Therefore, in addition to challenges from a numerical perspective, software packages that tackle this grand challenge, have to exploit the capabilities of state-of-the-art supercomputing architectures. In the past, simulations of seismic wave propagation used some of the largest supercomputers worldwide (e.g. [2, 3, 7–9, 17–19, 22, 29, 30]). However, only very few of the performed landmark-simulations coupled dynamic rupture propagation directly to seismic wave propagation (e.g. [10,17]). Taking the total number of simulation environments in the

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-41321-1_18

SCEC/USGS Spontaneous Rupture Code Verification Project [16] into account, a gap between latest physics-driven developments and HPC capabilities is visible. Reason is the required, high degree of algorithmic development, optimization and testing required to exploit all levels of parallelism offered by state-of-the-art supercomputing architectures [3].

The SeisSol software package[4] is the topic of this paper and uses, among other software (e.g. [1, 28]), the Discontinuous Galerkin (DG)-Finite Element Method (FEM) for spatial discretization. Together with the use of unstructured tetrahedral meshes and the Arbitrary high-order DERivatives (ADER) scheme in time, this allows for accurate discretization of fault systems, surface topography and material heterogeneities [13, 15, 23].

In this paper, we present various improvements of the software package SeisSol for the new Intel Knights Landing architecture (KNL). To maximize application performance, equaling shortest time-to-solution, our optimizations address KNL's major enhancements over the current architecture, code-named Knights Corner, by a) efficiently using both 512-bit wide vector processing units (VPU) per core, by b) leveraging the low-bandwidth DDR4 memory and the high-bandwidth in-package multi-channel DRAM (MCDRAM) by an out-of-core application memory management, and finally by c) balancing the on-die interconnect mesh-traffic for optimal throughput. In addition to our hardware-aware implementation, we demonstrate that advanced numerics and solvers are required for reduced time-to-solution. Here, SeisSol's computationally heavy wave propagation component was recently enhanced by a high performance Local Time Stepping (LTS) scheme to capture time step variations, commonly present in unstructured tetrahedral meshes [6]. Although the irregularities introduced by LTS normally contradict with the demands of modern and increasingly regular hardware architectures, such as KNL, we will demonstrate that our implementation is capable of running LTS efficiently on many-core processors with wide vector units.

2 The Knights Landing Architecture

The Intel Xeon Phi x200 processor family, based on the KNL architecture, is the successor of the Intel Xeon Phi coprocessor introduced in 2012. It is fully binary compatible with latest Intel Xeon processors code-named Haswell and Broadwell, e.g. Xeon E5v3 and E5v4, [5] and is the first chip that offers support for the AVX512F, AVX512CD, AVX512PF and the AVX512ERI instruction set extensions, which double the width of Intel Architecture's (IA) vector computing capabilities. AVX512F and AVX512CD instructions will be also available on future Intel Xeon processors and increase the number of programable 512-bit wide vector-registers to 32. In contrast to the first generation Xeon Phi coprocessor, KNL is intended to be operated in self-booted fashion and has therefore no need for a host processor. An overview of a KNL-based processor is depicted in Fig. 1.

[4] `https://github.com/SeisSol/SeisSol`, git-tag 201511 was used in this paper

[5] TSX instructions, however, are not considered to be legacy x86 instructions.

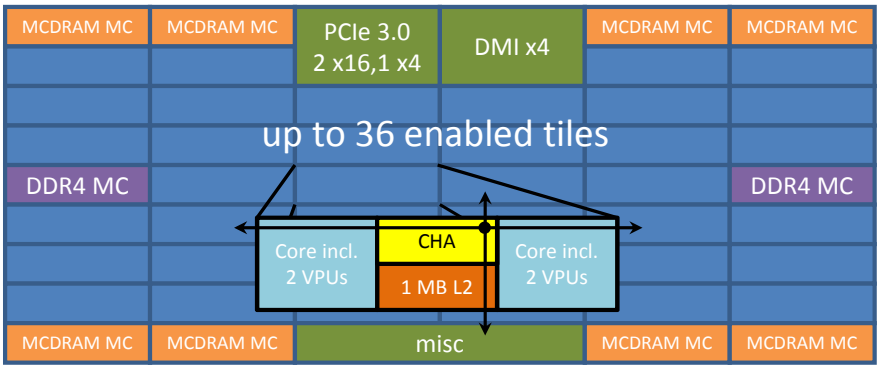

Fig. 1. Architectural overview of KNL: schematic die layout including the 2D-mesh of tiles and MCDRAM MC, DDR4 MC, IIO agents incl. a zoom into a tile.

The following descriptions are based on [26, 27], which disclosed many detailed architectural information of KNL.

KNL introduces many changes compared to KNC: up to 36 computing tiles (housing two cores with a shared L2 cache), 2 DDR4-2400 memory controllers (MC), 8 MCDRAM controllers (MCDRAM MC, accessing up to 16 GB in-package high-bandwidth memory) and a PCIe rootport with 36 PCIe3 lanes. All components are connected by a 2D mesh to ensure scalable communication within the die. Each DDR4 memory controller handles 3 channels with one DIMM each, allowing for up to 384 GB of system memory at 90 GB/s. The combined bandwidth of the eight high-bandwidth memory controllers exceeds 490 GB/s.

The computational heart of KNL is formed by an array of tiles. Each tile comprises two cores that share an 1 MB L2 cache and a Cache/Homing Agent (CHA). The latter one holds parts of a distributed tag directory which is used to maintain coherency across all L2 caches of all tiles. The cores are based on the Intel® Atom™ architecture code-named Silvermont [20], but offer many enhancements for HPC workloads. The most important one is the tighly coupled floating point unit (FPU) implemented by two 512-bit wide vector processing units (VPU), which support the aforementioned AVX512 instruction set extensions. Additionally, the cores feature larger L1 caches (32 KB each for data and instructions), more aggressive out-of-order execution and optimized support for huge pages. The core itself is two-issue-wide at instruction level (decode, retire) and supports up to six concurrent micro operations (2 VPU-, 2 memory-, 2 integer-operations). Thus, a single thread per core can utilize the full VPU-performance. The higher execution width is needed to optimally load the machine, e.g. to handle bursts after cache misses.

KNL's mesh can be operated in three different cluster modes which are selectable at boottime. As pointed out above, each tile holds a fraction of the distributed tag directory. The goal of KNL's cluster modes is to provide different

levels of affinity between the requesting tile, the tile which holds the corresponding tag entry, and the memory controllers. In the so-called ALL2ALL mode no affinities are enforced. This has the advantage that no explicit partitioning of memory controllers is required. However, this mode has also higher latencies as packages might travel through the entire chip. In QUADRANT mode the mesh is divided into four logical quadrants and an affinity between the tag directory and the memory controller is created by placing both in the same quadrant. Finally, Sub-NUMA-Clustering (or SNC4) is an extended version of the QUADRANT mode. Here, the four quadrants are exposed via NUMA domains to the OS such that applications can optimize memory access latencies even further.

KNL's memory subsystem is based on two different technologies. For capacity a 6-channel DDR4 is provided. For performance an up-to 16 GB large high-bandwidth in-package MCDRAM is provided. The MCDRAM can be used in different modes. The directly-mapped CACHE mode backs up the DDR4 memory. For applications that stay local or have a memory consumption of less than 16GB, this is a simple solution to get nearly all benefits from the high-bandwidth memory. Hence CACHE mode introduces an additional hierarchy, MCDRAM cache-misses add latency to the corresponding accesses. The second mode is the so-called FLAT mode. Here, the MCDRAM is exposed as an additionally NUMA domain in the physical address space and the programmer can explicitly request memory in this region by using close-to-metal libnuma or Intel's memkind[6] library. Note that the default memory in this mode is DDR4, such that the MCDRAM cannot get polluted by OS housekeeping. Finally, the HYBRID mode is a mixture of the CACHE and FLAT mode.

3 Computational Core

SeisSol solves the elastic wave equations, a linear system of partial differential equations with variable coefficients, in stress-velocity formulation:

$$q_t + A^{x_1} q_{x_1} + A^{x_2} q_{x_2} + A^{x_3} q_{x_3} = 0. \tag{1}$$

$q(\boldsymbol{x}, t) = (\sigma^{11}, \sigma^{22}, \sigma^{33}, \sigma^{12}, \sigma^{13}, \sigma^{23}, u^1, u^2, u^3)^T$ is the space-time-dependent vector of quantities containing the six-dimensional stress tensor and the particle velocities. The quantities q are functions of space $\boldsymbol{x} = (x_1, x_2, x_3)^T \in \mathbb{R}^3$ and time $t \in \mathbb{R}$. Here, the three normal stress components are given by σ^{11}, σ^{22} and σ^{33}, the three shear stresses by σ^{12}, σ^{13} and σ^{23}, and the three particle velocities in x_1-, x_2-, and x_3-direction by u^1, u^2 and u^3. The subscripts in (1) denote partial derivatives with respect to t and x_1, x_2, x_3. $A^{x_c}(\boldsymbol{x})$ are the three space-dependent Jacobian matrices (size 9×9) carrying the influence of the heterogeneous material [14]. Extensions of (1) might include source terms, viscoelasticity, anisotropy, or dynamic rupture physics [12, 14, 21, 23, 24].

We obtain the fully discrete formulation by applying the DG-machinery to (1) for space discretization and the ADER scheme in time [14, 21]. SeisSol uses

[6] `https://www.github.com/memkind/memkind`

static, unstructured tetrahedral meshes. Let Q_k (size $B_\mathcal{O} \times 9$) summarizes the per-element Degrees of Freedom (DOFs) for tetrahedral element k. The number of orthogonal basis functions $B_\mathcal{O}$ depends on the order of the overall scheme. In this work we present results for convergence rates $\mathcal{O} \in \{2, \ldots, 6\}$, leading to $B_2 = 4$, $B_3 = 10$, $B_4 = 20$, $B_5 = 35$ and $B_6 = 56$ basis functions. To advance an element k by its local time step, $t_k^{n_k+1} = t_k^{n_k} + \Delta t_k$, we compute the solution of SeisSol's time kernel, volume kernel and surface kernel.

Time: The time kernel predicts the evolution of the element-local DOFs within a time step. Following the Cauchy-Kowalewski procedure, we replace time derivatives by space derivatives and obtain:

$$\frac{\partial^{d+1}}{\partial t^{d+1}} Q_k(t_0) = -\sum_{c=1}^{3} \hat{K}^{\xi_c} \left(\frac{\partial^d}{\partial t^d} Q_k(t_0) \right) A_k^{\xi_c}. \tag{2}$$

$\hat{K}^{\xi_c}$ (size $B_\mathcal{O} \times B_\mathcal{O}$) are the three unique stiffness matrices, multiplied by the inverse, diagonal mass matrix in pre-processing. The stiffness matrices and the mass matrix are defined with respect to a reference element and in terms of the $\xi_1\xi_2\xi_3-$reference coordinate system. The matrices $A_k^{\xi_c}$ (size 9×9) are linear combinations of the Jacobians. We use the DOFs at the current time step $t_k^{n_k}$ as initial condition for the recursive procedure in (2): $\partial^0/\partial t^0 Q_k(t_0) = Q_k^{n_k}$. The time derivatives $\mathcal{D}_k = \partial^d/\partial t^d Q_k$ allow us to integrate the DOFs in time as required by the volume and surface kernel:

$$\mathcal{T}_k(t_0, \hat{t}, \Delta t) = \sum_{d=0}^{\mathcal{O}-1} \frac{\left(\hat{t} + \Delta t - t_0\right)^{d+1} - \left(\hat{t} - t_0\right)^{d+1}}{(d+1)!} \cdot \frac{\partial^d}{\partial t^d} Q_k(t_0). \tag{3}$$

Integration of the DOFs via (3) is valid in arbitrary intervals $[\hat{t}, \hat{t}+\Delta t]$ within the stability limits imposed by the CFL-condition. This translates to the condition $t_k^{n_k} \leq \hat{t} < \hat{t} + \Delta t \leq t_k^{n_k} + \Delta t_k$, where our element-local time step Δt_k satisfies the CFL-requirements. Depending on an element's LTS configuration, it stores different, permanent time data for read-only access by face-neighboring elements. Here, an element might store the derivatives $\mathcal{D}_k$, or add the full time integrated DOFs of the time step, $\mathcal{T}_k^{\text{full}} = \mathcal{T}_k(t_k^{n_k}, t_k^{n_k}, \Delta t_k)$, to a permanent buffer $\mathcal{B}_k$, or store both.

Volume: The volume kernel uses $\mathcal{T}_k^{\text{full}}$ and computes the net-effects of the volume integration for an entire, element-local time step Δt_k:

$$\mathcal{V}_k(\mathcal{T}_k^{\text{full}}) = \sum_{c=1}^{3} \tilde{K}^{\xi_c} \left(\mathcal{T}_k^{\text{full}}\right) A_k^{\xi_c}. \tag{4}$$

$\tilde{K}^{\xi_c}$ (size $B_\mathcal{O} \times B_\mathcal{O}$) are the three non-transposed stiffness matrices, multiplied with the inverse mass matrix in pre-processing. Analogue to the time derivative computation (2), $A_k^{\xi_c}$ are linear combinations of the Jacobians.

Surface: Our last kernel is the surface kernel, computing the surface integration of the fully discrete ADER-DG formulation. The surface kernel uses the

time integrated DOFs $\mathcal{T}_k^{\text{full}}$ of tetrahedron k and the time integrated DOFs $\mathcal{T}_{k_i}^{\text{part}}$ of the four face-neighboring tetrahedrons k_i. As discussed at the end of this section, $\mathcal{T}_{k_i}^{\text{part}}$ integrate face-neighboring derivatives $\mathcal{D}_{k_i}$ via (3), or directly use the buffer $\mathcal{B}_{k_i}$, containing one or multiple time integrated DOFs of the face-neighbor k_i. For a local face $i \in \{1, \ldots, 4\}$ of tetrahedron k, the kernel is given by:

$$\mathcal{F}_{k,i}\left(\mathcal{T}_k^{\text{full}}, \mathcal{T}_{k_i}^{\text{part}}\right) = \hat{F}^{-,i}\left(\mathcal{T}_k^{\text{full}}\right) \hat{A}_k^{-,i} + \hat{F}^{+,i,j_k(i),h_k(i)}\left(\mathcal{T}_{k_i}^{\text{part}}\right) \hat{A}_k^{+,i}. \quad (5)$$

$\hat{F}^{-,i}$ and $\hat{F}^{+,i,j,h}$ with $i, j \in \{1, \ldots, 4\}$ and $h \in \{1, 2, 3\}$ are the 52 unique flux matrices (size $B_{\mathcal{O}} \times B_{\mathcal{O}}$), multiplied by the inverse mass matrix in preprocessing. Here, the used indices j_k and h_k depend on the location of the elements' vertices in the reference element with respect to the shared face. As for the stiffness matrices and the mass matrix, the flux matrices are defined with respect to the unique reference element and thus shared among all elements. The matrices $\hat{A}_k^{-,i}$ (size 9×9) account for the element's own contribution to the numerical flux, while $\hat{A}_k^{+,i}$ (size 9×9) carry the contribution of the neighboring elements.

Update: By combining the individual kernels, we obtain the following two-step update scheme for an element-local time step $t_k^{n_k} \rightarrow t_k^{n_k+1}$:

$$Q_k^{*,n_k+1} = Q_k^{n_k} + \mathcal{V}_k - \sum_{i=1}^{4} \hat{F}^{-,i}\left(\mathcal{T}_k^{\text{full}}\right) \hat{A}_k^{-,i}, \quad (6)$$

$$Q_k^{n_k+1} = Q_k^{*,n_k+1} - \sum_{i=1}^{4} \hat{F}^{+,i,j_k(i),h_k(i)}\left(\mathcal{T}_{k_i}\right) \hat{A}_k^{+,i}. \quad (7)$$

Equation (6) summarizes all element-local contribution to the time step, while Eq. (7) accounts for the contribution of the face-neighboring elements.

Local Time Stepping: We use the Local Time Stepping (LTS) scheme introduced in [6] to account for heterogeneities in the CFL-imposed time step restrictions. This scheme trades some of the ADER scheme's flexibility, which in theory is able to advance each element with its optimal time step, for increased homogeneity. Here, we determine a fundamental time step equalling the global, minimal allowed time step of all elements. Afterwards, we assign every element to a cluster, such that it advances with an integer multiple of this fundamental time step. Considering the minimal, fundamental time step as Δt, the clustering reads as:

$$\mathcal{C}_1 = [\Delta t, r_1 \Delta t[,\ \mathcal{C}_2 = [r_1 \Delta t, r_1 r_2 \Delta t[,\ \ldots,\ \mathcal{C}_L = [r_1 \ldots r_{L-1} \Delta t, r_1 \ldots r_L \Delta t[. \quad (8)$$

With rates $r_l \in \mathbb{N}_{>1}$, we choose our L clusters to cover the entire interval of CFL-imposed time steps. In initialization all elements are assigned to their corresponding cluster. This work presents results for a clustering with fixed rates of $r_l = 2\ \forall l$. Further, the LTS scheme of [6] limits cluster dependencies and complex, worst-case memory handling by a normalization step, which lowers the time step of corner-case elements. All elements of a cluster advance in time with the cluster's lower time step limit. Global Time Stepping (GTS) is a special

case of our LTS scheme with a single cluster having rate $r_1 = \infty$. For GTS we store, in addition to the DOFs, the time integrated DOFs $\mathcal{T}_k^{\text{full}}$, computed for the element-local contributions in (6). These are then used in the update step (7) by face-neighboring elements.

In contrast, elements being in LTS-relation with at least one of their face-neighbors require a more complex handling. Here, an element might have to sum and store consecutive time integrated DOFs, obtained via (3), over multiple element-local time steps in a buffer $\mathcal{B}_k$ to feed face-neighboring elements with larger time steps. Conversely, elements having face-neighbors with smaller time steps store the time derivatives ($\mathcal{D}_k$), obtained using (2), which can then be evaluated by the face-neighbors in multiple evaluations of (7).

Summarizing, our LTS scheme is more challenging than GTS for the underlying hardware due to increased heterogeneity and memory requirements. In [6] we present full-machine results for a petascale, production character run on SuperMUC-2 (Haswell architecture). This run achieved 46 % of SuperMUC-2's HPL performance. Interpreting these results in terms of time-to-solution, rather than machine utilization, shows the real value of the LTS scheme. In the case of the rate-2, production character run, we reached a 4.1 × speedup over GTS.

4 Implementation

The discussion of the underlying ADER-DG discretization in SeisSol made clear that this algorithm is well suited for modern high-performance processors. The introduced update scheme requires dense compute capabilities (element-local operations in general) as well as high memory bandwidth for selected data structures ($\mathcal{B}_{k_i}$ and eventually $\mathcal{D}_{k_i}$ in the surface integral computation). In the upcoming subsections we will address how hardware features such as SIMD units and high-bandwidth memory can be leveraged to run high-order seismic simulations at high efficiencies. We discuss the following (co-)processors (Turbo mode being disabled):

HSX one Intel® Xeon® E5-2699v3 processor with 18 cores, 1.9 GHz at AVX-base frequency and up to 2.6 GHz Turbo frequency, 64 GB of DDR4-2133

KNC one Intel® Xeon Phi™ 7120A coprocessor in native mode with 61 cores, 1.24 GHz base and 1.33 GHz Turbo frequency, 16 GB of GDDR5, one core reserved for OS

KNL an Intel® Xeon Phi™ 7250 processor with 68 cores, 1.2 GHz AVX-base core-clock and 1.5 GHz all core Turbo frequency, 1.7 GHz mesh-clock, 16 GB MCDRAM@7.2 GT, 96 GB DDR4-2400, FLAT/(CACHE or QUADRANT), one core reserved for OS

4.1 Highly-Efficient Small Matrix Kernels

Small sparse and dense double precision matrix multiplication kernels form the computational back-bone of SeisSol. Single precision is possible but suffers from

accuracy issues for higher orders [5], we therefore restrict ourselves to double precision in this work. As pointed out in previous work [5,7,17], the best strategy is to generate optimal code for these kernels. After an auto-tuning exercise, we found out that a fully dense backend is the best choice on KNL. Note that also on latest Intel Xeon processors (HSX) the sparse/dense tuning achieves only between 12% (order 2) and 1.5% (order 6) improvement with respect to time-to-solution. For the remainder of this section, we rely on regular BLAS notation: $C = \alpha A \cdot B + \beta C$, $C \in \mathbb{R}^{M \times N}$, $A \in \mathbb{R}^{M \times K}$ and $B \in \mathbb{R}^{K \times N}$. lda, ldb and ldc define the length of the leading memory dimension of each matrix, and therefore lda $\geq M$, ldb $\geq K$ and ldc $\geq M$. Since we only need the simple cases of $\alpha = 1$ and $\beta \in \{0, 1\}$, we do not discuss the efficient integration of arbitrary α and β values into our kernels. A generalized version ($N \neq 9$) of the presented code generation approach is used in the back-end of the LIBXSMM open source project[7]. This library is already used in other scientific applications (e.g. CP2K [4] or Nek5000 [25]) which demand small matrix multiplications as well.

As we have discussed the implementation of SeisSol's kernels on older Intel architectures in detail in [5], we only focus on KNL in this article. Since KNL has 32 architectural registers available and we know that $N = 9$ holds always true, we decided to work in all cases on all columns of B and C simultaneously. A naive implementation might load 8 rows of column k of A into a register and then perform 9 FMA instructions, which broadcast the kth row of all 9 columns of B on the fly. After having processed all columns of A and rows of B, we would hold a 8×9 sub-matrix of C in 9 accumulator registers which are stored back to all 9 columns of C. However, such a kernel would suffer many instruction level dependencies which block efficient execution. An optimal AVX512 implementation needs to consider therefore two points: a) eliminating dependencies by software pipelining to reduce pressure on micro-op level and b) ensuring smallest possible instructions to reduce pressure on the frontend.

The problem of a) is twofold. First, the innermost kernel consists of 9 FMA instructions which presumably run in throughput scenarios in 4.5 cycles as there are 2 VPUs per out-of-order core with a latency of 6 cycles. This puts high pressure on the core as the same nine registers (e.g. `zmm23-31`) will be reused in the next iteration of the microkernel. As a solution we introduce a second temporary accumulator for C, `zmm14-22`, which is used in every other iteration. This ensures that the same register is only reused after at least 9 cycles. Before storing back to C we need to merge `zmm23-31` and `zmm14-23`, however the overhead in case of a larger K is minor. Second, we pipeline the loads of rows per column k of A to get them as early as possible into the core's pipeline. This is easily doable as registers `zmm0-13` are still unused: we implement a 6-register ring-buffer of A column-vectors.

Issue b), ensuring smallest possible instructions, is more problematic since we cannot afford to re-structure our data as it is normally done for large DGEMMs. We therefore have strided accesses (offset is ldb times 8), when reading B in the

[7] `https://github.com/hfp/libxsmm`

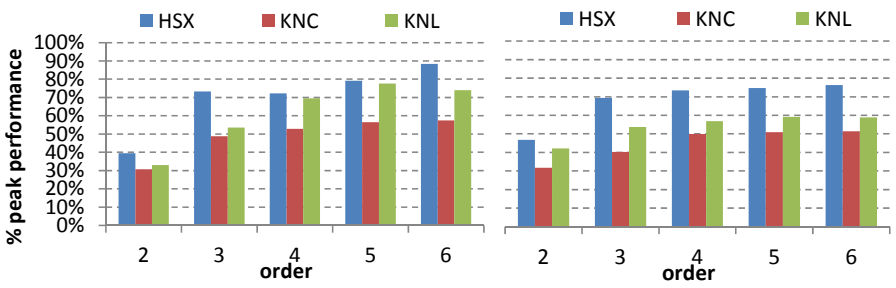

Fig. 2. Standalone matrix kernel performance running out of a hot L1 cache for HSX, KNC and KNL. Left: kernel performance for $B_{\mathcal{O}} \times 9 \times B_{\mathcal{O}}$ matrix multiplication shapes; right: kernel performance for $B_{\mathcal{O}} \times 9 \times 9$ matrix multiplication shapes.

FMA-fused broadcast. If the offset exceeds 128 bytes, the length of the FMA instruction increases from 7 to 11 bytes which puts avoidable pressure on the fetch and decoder units. However, the instruction size can be fixed to 8 byte per FMA if the x86 SIB scale-index-base (SIB) addressing mode is utilized. Since we have spare general purpose registers, we can express the 9 column streams of B by SIB with different base registers (to the first, fourth and seventh column of B) and multiples ($\{1,2,4,8\}$) of ldb. Every 128th k we need to increase these pointers by 128 to remain in the one-byte offset range. In fact 128 elements in k-direction are possible as the AVX512 FMA instructions use a special encoding for the memory offest: they scale the offset value by the datatype size. For example, if the encoded offset is 55, then the offset used during the memory access is $55 \cdot 8 = 440$ (assuming double precision numbers).

Fig. 2 compares the performance for the most often used kernel operations in SeisSol running single-threaded on HSX, KNC and KNL. HSX numbers are taken from [5]. For both operator shapes ($M \times N \times K$), $B_{\mathcal{O}} \times 9 \times B_{\mathcal{O}}$ and $B_{\mathcal{O}} \times 9 \times 9$, KNL clearly outperforms its previous generation (KNC). For $B_{\mathcal{O}} \times 9 \times B_{\mathcal{O}}$ nearly HSX performance is achieved. The governing reason for the lower performance compared to HSX is KNL's two-issue-wide pipeline: all instructions which are not FMA instructions reduce the attainable FLOPS peak. Since these occur relatively more often for the $B_{\mathcal{O}} \times 9 \times 9$ operations, its performance is accordingly lower on KNL than the performance of the $B_{\mathcal{O}} \times 9 \times B_{\mathcal{O}}$ shapes.

4.2 Out-of-Core Time Kernel

SeisSol's wave propagation solver is implemented by two macro-kernels: the regular time kernel fused with the element-local volume kernel and element-local part of the surface kernel (6), and the contribution of the face-neighboring elements (7). In the case of high-order simulations the access frequency to Q_k, $\mathcal{B}_k$ or $\mathcal{D}_k$ and the element-local $A_k^{\xi_c}$, $\hat{A}_k^{-,i}$ in the computation of the local contributions is very low, as the data causing the majority of the compute ($\hat{K}^{\xi_c}$, $\tilde{K}^{\xi_c}$, $\hat{F}^{-,i}$ and temporary buffers) can be cached in each tile. However, gathering the neighboring contributions, $\mathcal{B}_{k_i}$ or $\mathcal{D}_{k_i}$, requires significantly more bandwidth than $\hat{A}_k^{+,i}$ for

order	Q_k	$\mathcal{B}_k, \mathcal{D}_k$	$A_k^{\xi_c}, \hat{A}_k^{-,i}, \hat{A}_k^{+,i}$	$\hat{K}^{\xi_c}, \tilde{K}^{\xi_c}, \hat{F}^{-,i}, \hat{F}^{+,i,j,h}$
2	MCDRAM	MCDRAM	MCDRAM	MCDRAM
3	MCDRAM	MCDRAM	MCDRAM	MCDRAM
4	DDR4	MCDRAM	MCDRAM	MCDRAM
5	DDR4	MCDRAM	DDR4	MCDRAM
6	DDR4	MCDRAM	DDR4	MCDRAM

Table 1. Placements for all orders and the different data structures of SeisSol; DDR4/MCDRAM denotes if a particular data structure is placed in DDR4/MCDRAM.

higher orders as they are bigger but have the same access frequency. These access patterns allow to overcome size limitations of the 16 GB MCDRAM by placing the 'slow-running' data structures in DDR4. Therefore, in FLAT mode and for higher order runs, we store $\mathcal{B}_k$ and/or $\mathcal{D}_k$ of every element into MCDRAM on the fly via the memkind library when computing them. As both memory types are seamlessly integrated into the architecture, we simply change the place of allocation, but not our macro-kernels. Thus pointers to $\mathcal{B}_k$ and/or $\mathcal{D}_k$ reference memory physically stored in MCDRAM whereas $A_k^{\xi_c}, \hat{A}_k^{-,i}, \hat{A}_k^{+,i}, Q_k$ reside in the DDR4 portion of the address space for orders $\mathcal{O} = 5$ and $\mathcal{O} = 6$. Additionally, we hold unique matrices, $\hat{K}^{\xi_c}, \tilde{K}^{\xi_c}, \hat{F}^{-,i}, \hat{F}^{+,i,j,h}$, including the 48 flux matrices required for neighboring elements' contribution to the surface kernel (7), in MCDRAM as well, as we expect local L2 cache evicts for higher orders. For lower orders, two to four, the bandwidth requirements of SeisSol for the element local matrices and Q_k increase. We therefore allocate more data structures in MCDRAM. In fact, for orders $\mathcal{O} = 2$ and $\mathcal{O} = 3$, all important data structures are placed in MCDRAM. Table 1 summarizes the used placements, when running on KNL in FLAT mode.

4.3 Optimizing the Mesh Traffic and Prefetching

KNL's last level cache (LLC) is not a shared cache level as it is implemented by a 2D mesh of up to 36 1 MB large slices of L2 caches, c.f. Sect. 2. These slices are kept coherent by a distributed tag directory in each tile's CHA. As we pointed out in the last section, for higher orders than four, the 48 flux matrices $F^{+,i,j,h}$ approach (500 KB for order five) or even exceed the size (1.5 MB for order $\mathcal{O} = 6$) of one tile's L2 cache. This can negatively effect the performance of (7) for two reasons: a) especially for order $\mathcal{O} = 6$ this results into a high rate of CHA-to-CHA communication as the unstructured mesh causes unstructured accesses to the flux matrices b) the hardware prefetcher cannot pick-up the unstructured accesses. Keeping the last section in mind, we know that we still have plenty of MCDRAM bandwidth available in higher orders. Therefore, we place several copies, one per two tiles, in MCDRAM. This ensures that the mesh traffic gets equally distributed and the access latency may not be limited by one CHA in the entire mesh holding the directory entries for one particular flux matrix. Additionally, we are using modified matrix kernel operations in (7),

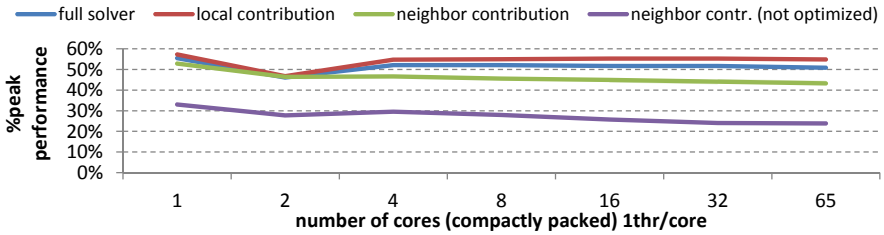

Fig. 3. Scaling of a setup with LOH.1 characteristics (c.f. Sect. 5) on KNL using global time stepping. Shown is the separated performance of the element local contribution (6) and the contribution of the face-neighboring elements (7) and the combined full solver for order $\mathcal{O} = 6$ (measured by a performance proxy application for single-node SeisSol executions with errors of less than 1%). Additionally, we show the scaling of the neighboring elements' contribution to the surface kernel without our optimization for KNL's mesh and distributed LLC.

which allow for prefetching the flux matrix required for the next face-neighbor's contribution as well as the next $\mathcal{B}_{k_i}$ or $\mathcal{D}_{k_i}$. For best performance these prefetches are widely scattered throughout all eight matrix operations.

The effects of these tweaks are depicted in Fig. 3 when running a setup with LOH.1 characteristics, c.f. Sect. 5, using order $\mathcal{O} = 6$ in FLAT/QUADRANT mode on KNL. The plot shows scaling curves for the local part (6), the neighbor element's contribution (using no optimization and all optimization discussed above), and SeisSol's overall scaling using the optimized neighboring contribution (7). Its aforementioned performance tweaks roughly double the performance of (7) and result in nearly perfect scaling. For all operations the biggest scaling drop occurs when moving from one to two cores. The reason for this is the shared L2 cache per tile which allows for reading one line per cycle and writing a half line per cycle. This effects the performance of (7) more severe, since more data (flux matrices, time integrated DOFs/time derivatives, flux solvers) are read per element as in case of the element-local integrations. As for order $\mathcal{O} = 6$ the local part (6) takes up roughly 70% of SeisSol's total runtime, the overall scaling follows the scaling of the (6). The full solver's performance is only slightly affected by the lower performance of (7).

5 Scenarios

In this section we evaluate the performance of three different scenarios. The first scenario, LOH.1, is a wave propagation benchmark, the second setting simulates seismic wave propagation in the volcano Mount Merapi, while the last configuration is a multi-physics dynamic rupture simulation of the 1992 Landers earthquake.

Our performance comparisons are carried out on a socket-to-socket basis for two reasons: a) the power per KNL-socket is only ≈50% higher than for a single-socket HSX and b) Intel's reference platforms for KNL and HSX pack 4 sockets

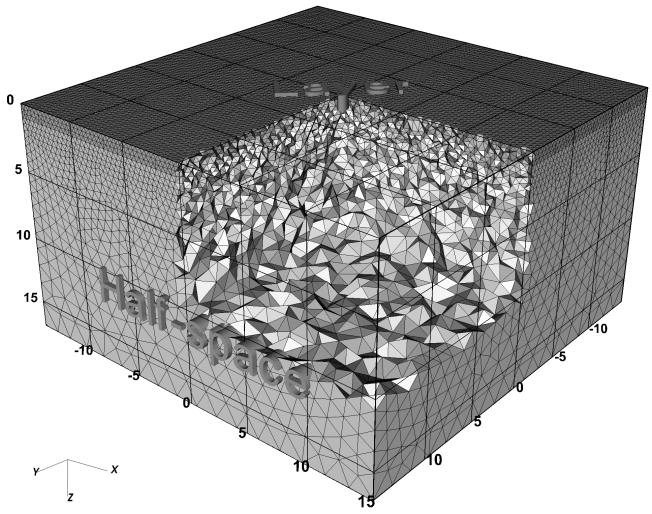

Fig. 4. Illustration of the Layer Over Half-space (LOH.1) setup. Shown is the domain $\Omega = [-15\,\text{km}, 15\,\text{km}]^2 \times [0, 17\,\text{km}]$. The upper part of the domain is covered by the 1 km thick layer (dark gray) and the remainder by the half-space (gray). The structure of the mesh is illustrated by removing the elements in $[0, 15\,\text{km}]^2 \times [0, 10\,\text{km}]$.

of each into 2U of rack-space. Furthermore, in case of KNL the socket power includes also MCDRAM power, therefore for a single-socket comparison roughly the same amount of energy is spent in the actual CPU. Additionally, SeisSol is known to run large-scale equivalents of the used Mount Merapi and Landers setups well to more than 100,000 cores [6, 7, 17].

5.1 LOH.1

The Layer Over Half-space benchmark [11] consists of two different material regions. The higher resolved layer is located at the flat surface and reaches 1 km deep into the computational domain. We use material parameters $\rho = 2600\,\text{kg/m}^3$, $\lambda = 20.8\,\text{GPa}$, and $\mu = 10.4\,\text{GPa}$ for the layer. The half-space covers the remaining part of the computational domain. Here, we use material parameters $\rho = 2700\,\text{kg/m}^3$, $\lambda = 32.4\,\text{GPa}$, and $\mu = 32.4\,\text{GPa}$. Fig. 4 illustrates the 386,518-element mesh of the LOH.1 benchmark. The faces of the tetrahedral elements are aligned to the interface of the layer and the half-space, and are aligned to the boundary of the computational domain. Boundary conditions are free-surface for the top of the computational domain ($z = 0$) and outflow everywhere else. We use a point dislocation at $(0, 0, 2\text{km})$ as seismic source.

The upper plot of Fig. 5 depicts the speed-up over global time stepping (GTS), executed on HSX, with respect to time-to-solution for the LOH.1 scenario. In terms of FLOPS, this translates into roughly 1.2 TFLOPS of raw performance on KNL which is $\approx 4\times$ more than on HSX. However, we have to

keep in mind, that we are using different sparse/dense switches for each operator on HSX, KNC and KNL (see Ch. 4.1, [7]). Therefore, the only fair comparison is time-to-solution. In this measure, KNL achieves a speed-up of 2.1 - 3.4 $\times$ depending on the chosen order of convergence in global time stepping (GTS) runs and baseline architecture (upper plot of Fig. 5). We pad Q_k, $\mathcal{B}_k$ and $\mathcal{D}_k$ in their respective data structures on a per-element basis. On HSX we pad to the next 32-byte boundary and on KNL/KNC to the next 64-byte boundary and store $A_k^{\xi_c}$ dense on KNL, therefore the lower speed-up for lower orders (two to four) is expected. Here, the execution is memory bandwidth bound. In the case of $\mathcal{O} = 2$, KNL/KNC have to move roughly twice as much data as HSX. How heavily these low orders are bandwidth bound can also be seen from the $\approx$3$\times$ faster computations resulting from execution out of MCDRAM. For higher orders the MCDRAM-benefit is measurable, but much smaller. It is worthwhile noting that the LOH.1 benchmark fits into MCDRAM for every order. At order 6 all data structures consume $\approx 6\,\mathrm{GB}$. Therefore it does not matter if the MCDRAM is used in the explicit FLAT or the implicit CACHE mode. When enabling rate-2 local time stepping (LTS) in SeisSol, a theoretical speed-up of 2.8 $\times$ over GTS can be achieved. For higher orders HSX can achieve close to 95% of this value and KNL can reproduce 95% of HSX's LTS speed-up. The slightly lower speed-up is due to the cluster sizes and their distribution: the first and most often updated cluster contains less than 0.5 % of all elements whose calculations have to be parallelized across 67 cores on KNL instead of 18 on HSX. Nevertheless, when comparing to the HSX GTS baseline, KNL is able to execute the LOH.1 benchmark up to 7.7 $\times$ faster.

5.2 Mount Merapi

Our second setting simulates seismic wave propagation in the volcano Mount Merapi. Except for the smaller mesh, now having 1,548,496 tetrahedral elements, this setting is identical to the one used in [6, 7]. The origin $(0, 0, 0)$ of our setup is located at mean sea level below Mount Merapi's peak. For elements inside the volcano, being in the sphere with radius 5.1 km and center $(4\,\mathrm{km}, 0, 0)$, we use the material settings $\rho = 2400\,\mathrm{kg/m^3}$, $\lambda \approx 3.3\,\mathrm{GPa}$ and $\mu \approx 4.7\,\mathrm{GPa}$. All remaining elements have paramters $\rho = 2000\,\mathrm{kg/m^3}$, $\lambda \approx 2.3\,\mathrm{GPa}$ and $\mu \approx 2.4\,\mathrm{GPa}$. Two different characteristic lengths for element sizes are used inside and outside the volcano.

Fig. 6 illustrates three different clusters for rate-2 clustering ($r_l = 2\ \forall l$ in (8)). From the left to the right, we see the elements of clusters $C_2 = [2\Delta t, 4\Delta t[$, $C_3 = [4\Delta t, 8\Delta t[$ and $C_4 = [8\Delta t, 16\Delta t[$. The colors of the elements correspond to the element-local CFL-imposed time step. Boundary conditions are free-surface at the surface and outflow everywhere else. The faces of our tetrahedral elements are aligned to the surface topography, the material contrast and the spherical shape of the outflow boundary. We use a double-couple point source approximation at $(0, 0, 0)$ as seismic source in the Mount Merapi setup.

Compared to the LOH.1 setup, the larger mesh allows us to analyze our out-of-core implementation in more detail. Fig. 7 depicts the time-to-solution

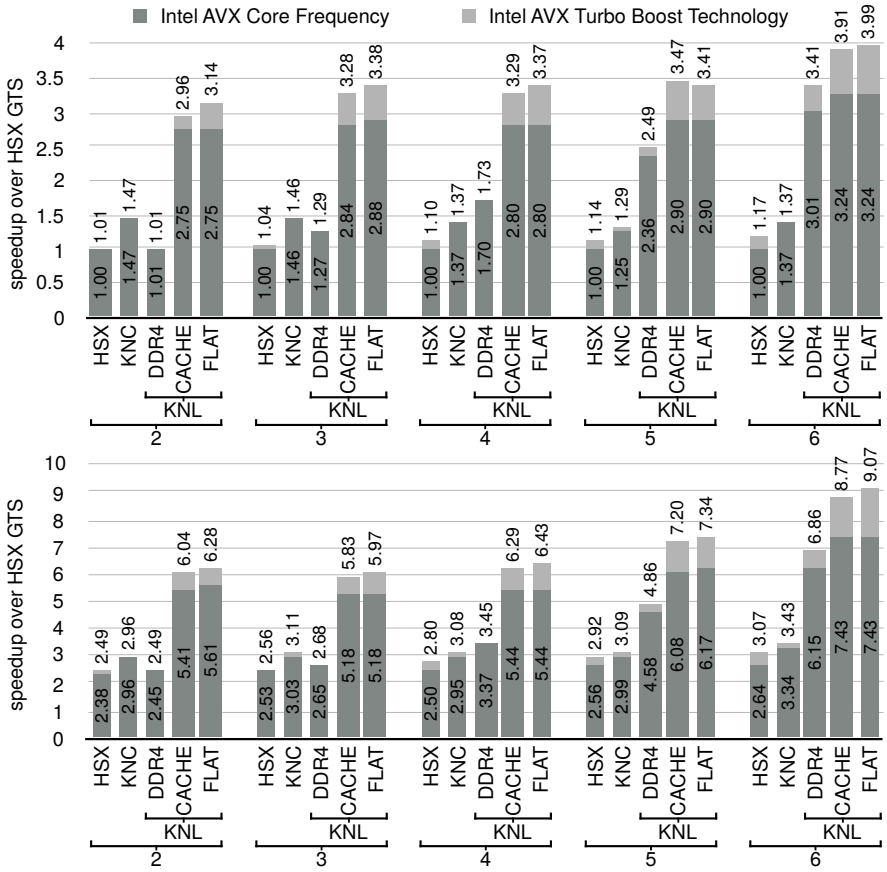

Fig. 5. Normalized time-to-solution speed-up in the LOH.1 scenario for HSX, KNC and KNL and orders 2-6. Upper plot: global time stepping, Lower plot: rate-2 local time stepping.

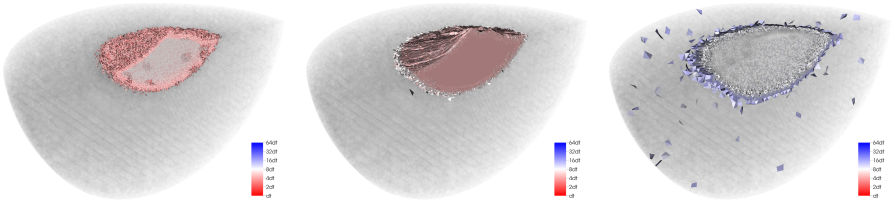

Fig. 6. Three LTS clusters of the Merapi configuration. Shown are, from left to right: $\mathcal{C}_2 = [2\Delta t, 4\Delta t[$, $\mathcal{C}_3 = [4\Delta t, 8\Delta t[$, $\mathcal{C}_4 = [8\Delta t, 16\Delta t[$.

when executing the Mount Merapi scenario, here rate-2 LTS can gain $4\times$ in theory with respect to time-to-solution. The increased mesh size is reflected by KNC's performance results: due to lack of memory we can not execute the

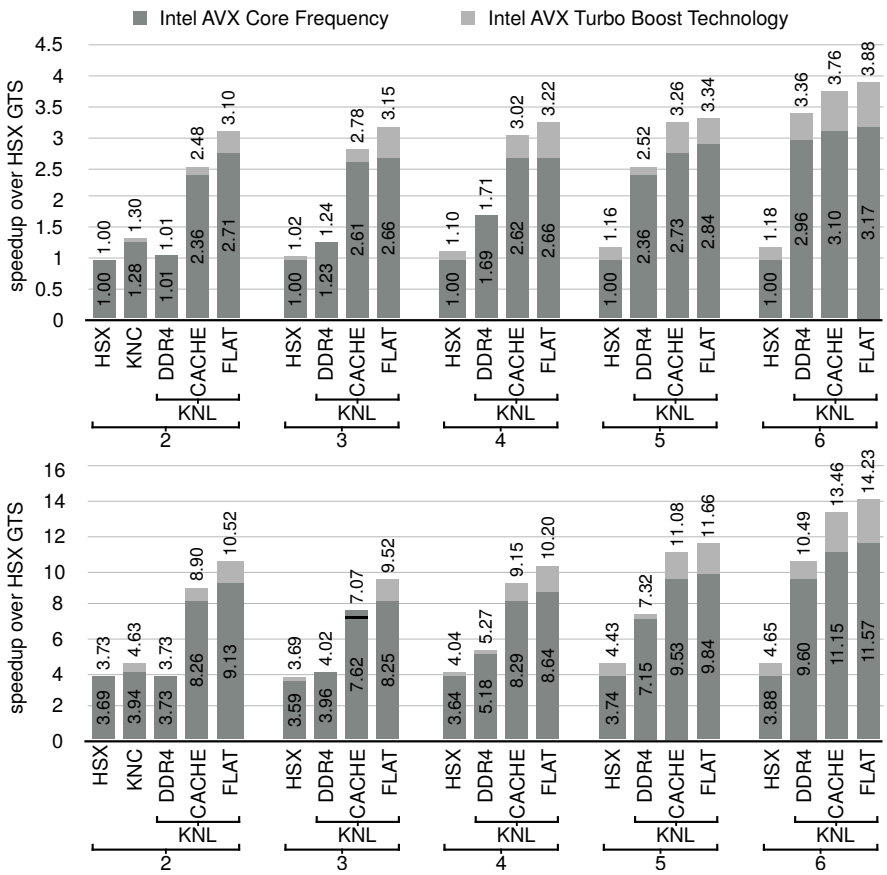

Fig. 7. Normalized time-to-solution speed-up in the Mount Merapi scenario for orders 2-6 and (co)processors HSX, KNC and KNL over HSX global time stepping. Upper plot: global time stepping, Lower plot: rate-2 local time stepping.

simulation for orders larger than two. In contrast, on KNL this limitation is no longer present. As the Merapi scenario achieves LOH.1-comparable speed-ups over HSX in FLAT, our out-of-core implementation is not limited by KNL's DDR4 bandwidth, e.g. for order $\mathcal{O} = 6$ the total consumed memory is 25 GB with 7.3 GB used in MCDRAM. For LTS the total memory consumption increases to 30 GB and 11 GB of used MCDRAM. While in GTS every element k only stores a buffer B_k for read-only access by face neighbors, an element k in LTS configurations might have to store buffers $\mathcal{B}_k$, or derivatives $\mathcal{D}_k$, or both $\mathcal{B}_k$ and $\mathcal{D}_k$. Even the software-transparent CACHE mode of the MCDRAM helps a lot compared to a pure DDR4 execution as its performance is always within 10% of the manually optimized FLAT mode implementation. As a bottom line we can conclude that KNL can execute the Mount Merapi scenario up to 12.1 $\times$ faster than the HSX GTS baseline.

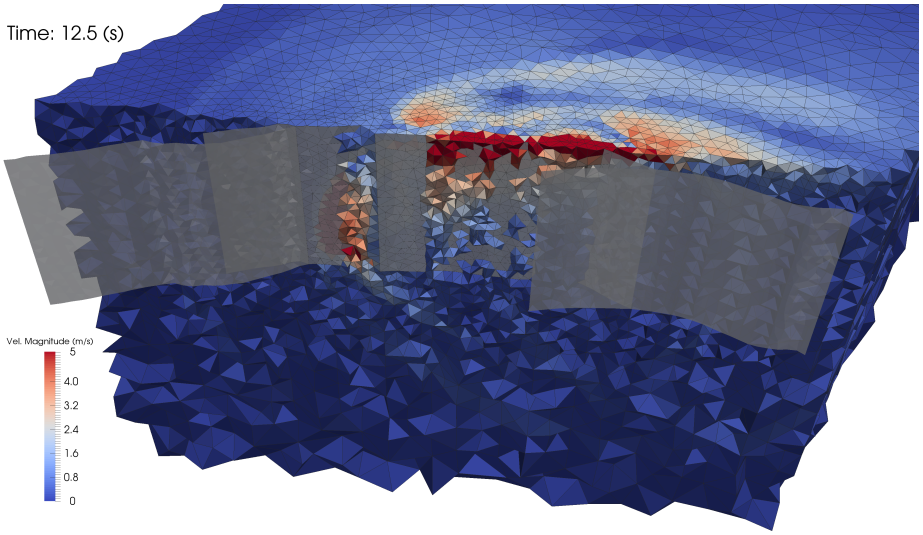

Fig. 8. Wave field of the 1992 Landers scenario after 12.5 s of simulated time. Shown is the fault system with a subsection of the unstructured tetrahedral mesh.

5.3 1992 Landers

The 1992 Landers setup is similar to the large-scale, production configuration of [17]. However, in this work we only use a total of 466,574 tetrahedrons to discretize the spatial domain. A higher mesh resolution is used to represent the geometry of the fault system and the topography. We solve dynamic rupture physics for faces aligned to the fault system, depicted in Fig. 8. Effectively, we replace our Riemann solver, used in the surface kernel of Sect. 3, with a formulation explicitly enforcing a Godunov state, which satisfies a certain friction law [23]. Boundary conditions are free-surface at the surface and outflow everywhere else.

Material parameters in the domain are discretized using a one-dimensional, layered velocity profile. This velocity profile leads to gradually increasing wave speeds with increasing depth. The 1992 Landers setup uses global time stepping and orders 2-6 for the seismic wave propagation component. For the dynamic rupture computations a single quadrature point in time and multiple quadrature points in space are used [17]. Note that our computational core supports dynamic rupture physics only in GTS execution. While our considerations for the LTS wave propagation component in [6] directly translate to dynamic rupture elements, extensive benchmarking is required to validate local time stepping in dynamic rupture workloads. Here, one can either decide to follow the LTS approach of the scheme in [6] directly and perform a minimal impact normalization only. Other options could enforce neighboring dynamic rupture elements to have the same time step or enforce a shared, minimal time step for all elements with dynamic rupture faces. As in case of the LOH.1 scenario, all data structures

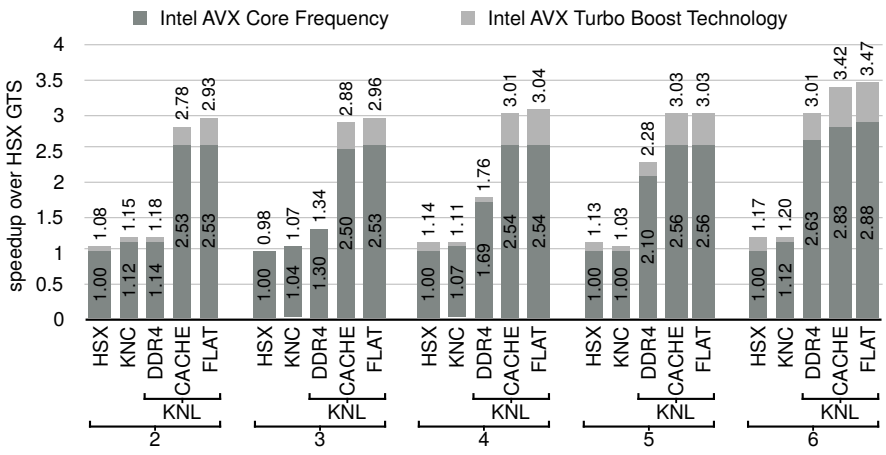

Fig. 9. Normalized time-to-solution speed-up over HSX for KNC and KNL and orders 2-6 when simulating the 1992 Landers scenario using global time stepping.

would easily fit into MCDRAM any time as the total memory consumption at order 6 is 7.1 GB.

The GTS performance of the 1992 Landers setup is provided in Fig. 9. As this is a multi-physics scenario, we expect slightly lower performance than for the earlier pure wave propagation runs on a many-core processor. This is due to the fact that the dynamic rupture portion of the solver requires high scalar performance. Here, KNL's increased single-thread performance becomes visible. KNL reassembles more than 92% of the pure wave propagation speed-up over HSX whereas the previous generation KNC chip is only able to attain 83%. This results into a relative performance which is comparable to HSX. KNL's time-to-solution speed-up for executing the 1992 Landers earthquake simulations is 2.5 - 2.9 × depending on the chosen order.

6 Conclusion

In this article, we presented a holistic optimization of SeisSol, a multi-physics simulation package for seismic simulations, which tightly couples seismic wave propagation, and dynamic rupture processes. First, we presented a deep-dive into KNL's architectural features and their challenges and opportunities for high-performance software. After a brief recapitulation of SeisSol's mathematical background, we discussed in detail how to exploit KNL's two VPUs per core efficiently and to leverage both memory subsystems for a novel out-of-core implementation in SeisSol's high-order wave propagation solver. The KNL-optimized implementation was evaluated for three different scenarios with distinct challenges and sizes. In case of global time stepping runs, KNL was able to outperform its predecessor, KNC, by 2.9 × and the current most powerful Intel Xeon processor, E5v3, by more than 3.4 ×. Even more important, in contrast to

KNC, KNL can maintain its speed-up over the E5v3 also when boosting time-to-solution via local time stepping, resulting into a more than 12.1 × speed-up when comparing against global time stepping runtimes on Intel Xeon E5v3. Up to 3.1 × faster execution on KNL is possible when taking local time stepping runtimes as a baseline. In summary, our results have demonstrate that for best time-to-solution we must not only rely on performance engineering (increasing achieved FLOPS) but also investments in algorithmic design achieving best asymptotic complexity (increasing the ratio of science/FLOP).

References

1. Mondher Benjemaa et al. 3-d dynamic rupture simulations by a finite volume method. *Geophysical Journal International*, 2009.
2. Jacobo Bielak et al. Parallel octree-based finite element method for large-scale earthquake ground motion simulation. *Computer Modeling in Engineering and Sciences*, 2005.
3. Jacobo Bielak et al. The shakeout earthquake scenario: Verification of three simulation sets. *Geophysical Journal International*, 2010.
4. Urban Borstnik et al. Sparse matrix multiplication: The distributed block-compressed sparse row library. *Parallel Computing*, 40(56), 2014.
5. Alexander Breuer et al. High-order ader-dg minimizes energy- and time-to-solution of seissol. In *Proceedings of ISC15*.
6. Alexander Breuer et al. Petascale local time stepping for the ader-dg finite element method. Accepted at IPDPS15, available online: `http://www5.in.tum.de/~bader/preprint/ipdps16_for_isc16review.pdf`, password: `isc16_review`
7. Alexander Breuer et al. Sustained petascale performance of seismic simulations with seissol on supermuc. In *Proceedings of ISC2014*. PRACE ISC Award 2014.
8. Laura Carrington et al. High-frequency simulations of global seismic wave propagation using specfem3d_globe on 62k processors. In *Proceedings of SC08*.
9. Yifeng Cui et al. Physics-based seismic hazard analysis on petascale heterogeneous supercomputers. In *Proceedings of SC13*.
10. Yifeng Cui et al. Scalable earthquake simulation on petascale supercomputers. In *Proceedings of SC10*.
11. Steven M Day et al. Tests of 3d elastodynamic codes: Final report for lifelines project 1a02. *Pacific Earthquake Engineering Research Center*, 2003.
12. Josep de la Puente et al. An arbitrary high-order discontinuous galerkin method for elastic waves on unstructured meshes-iv. anisotropy. *Geophysical Journal International*, 169(3), 2007.
13. Josep de la Puente et al. Dynamic rupture modeling on unstructured meshes using a discontinuous galerkin method. *Journal of Geophysical Research: Solid Earth*, 2009.
14. M. Dumbser and M. Käser. An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes – II. The three-dimensional isotropic case. *Geophysical Journal International*, 167(1), 2006.
15. Michael Dumbser et al. An arbitrary high-order discontinuous galerkin method for elastic waves on unstructured meshes—ii. the three-dimensional isotropic case. *Geophysical Journal International*, 2006.
16. RA Harris et al. The scec/usgs dynamic earthquake rupture code verification exercise. *Seismological Research Letters*, 2009.

17. Alexander Heinecke et al. Petascale high order dynamic rupture earthquake simulations on heterogeneous supercomputers. In *Proceedings of SC14*. Gordon Bell Finalist.
18. Tsuyoshi Ichimura et al. Implicit nonlinear wave simulation with 1.08 t dof and 0.270 t unstructured finite elements to enhance comprehensive earthquake simulation. In *Proceedings of SC15*.
19. Tsuyoshi Ichimura et al. Physics-based urban earthquake simulation enhanced by 10.7 blndof× 30 k time-step unstructured fe non-linear seismic wave simulation. In *Proceedings of SC14*.
20. Intel Corporation. Intel(R) 64 and IA-32 Architectures Optimization Reference Manual. January 2016.
21. Martin Käser et al. An arbitrary high-order discontinuous galerkin method for elastic waves on unstructured meshesiii. viscoelastic attenuation. *Geophysical Journal International*, 168(1), 2007.
22. Dimitri Komatitsch et al. High-order finite-element seismic wave propagation modeling with mpi on a large gpu cluster. *Journal of Computational Physics*, 2010.
23. Christian Pelties et al. Three-dimensional dynamic rupture simulation with a high-order discontinuous galerkin method on unstructured tetrahedral meshes. *Journal of Geophysical Research: Solid Earth*, 2012.
24. Christian Pelties et al. Verification of an ader-dg method for complex dynamic rupture problems. *Geoscientific Model Development Discussion*, 6, 2013.
25. Jaewook Shin et al. Speeding up nek5000 with autotuning and specialization. In *Proceedings of ICS10*.
26. Avinash Sodani. Knights Landing (KNL): 2nd Generation Intel(R) Xeon Phi(TM) Processor. In *Hotchips-2015*.
27. Avinash Sodani et al. Knights Landing (KNL): 2nd Generation Intel(R) Xeon Phi(TM) Processor. *IEEE Micro, Hot Chips Special Issue, to appear*, March 2016.
28. Josué Tago et al. A 3d hp-adaptive discontinuous galerkin method for modeling earthquake dynamics. *Journal of Geophysical Research: Solid Earth*, 2012.
29. Tiankai Tu et al. From mesh generation to scientific visualization: An end-to-end approach to parallel supercomputing. In *Proceedings of SC06*.
30. Lucas C Wilcox et al. A high-order discontinuous galerkin method for wave propagation through coupled elastic–acoustic media. *Journal of Computational Physics*, 2010.