DEEP LEARNING HARDWARE ACCELERATES FUSED DISCONTINUOUS GALERKIN SEISMIC SIMULATIONS

Alexander Heinecke

Parallel Computing Lab
Intel Labs
USA

April 23th 2018

Legal Disclaimer & Optimization Notice

This is a Cross-Organization Team Effort!

Alexander Breuer (UCSD)

Yifeng Cui (UCSD)

Josh Tobin (UCSD)

Junyi Qui (UCSD)

Charles Yount (Intel SSG DRD)

Alexander Heinecke (Intel Labs)

What are Deep Learning’s Compute Requirements

- Deep Learning (CNN, RNN, LSTM) is becoming the next “Killer” App
- It exhibits very regular compute patterns which can be performed at lower precision
  - CNN even work with a couple of bits
  - RNN/LSTM require a bit higher precision, but by far not FP64
- The most common kernel is GEMM and convolution which both map to long chains of many independent inner products
- Convolutions even exhibit greater spatial and temporal locality than GEMM
- Intel (->QFMA) and Nvidia (->TensorCore) have announced special function units to speed up matrix multiplications

What do we have to do to harvest this for HPC

- Mixed Precision
  - leverage high amount of lower precision flops
    - Running preconditioner with very low precision
    - Evaluation if solvers can be run with FP32 instead of FP64
- Regular/GEMM type of compute in combination with low BW requirements
  - High-order Methods
    - GEMM-like kernels
    - Good locality of order is chosen sufficiently high
    - May become BW bound when being used in implicit fashion ☹
  - Additional locality: Fused Simulations + High Order Methods (more in this presentation)

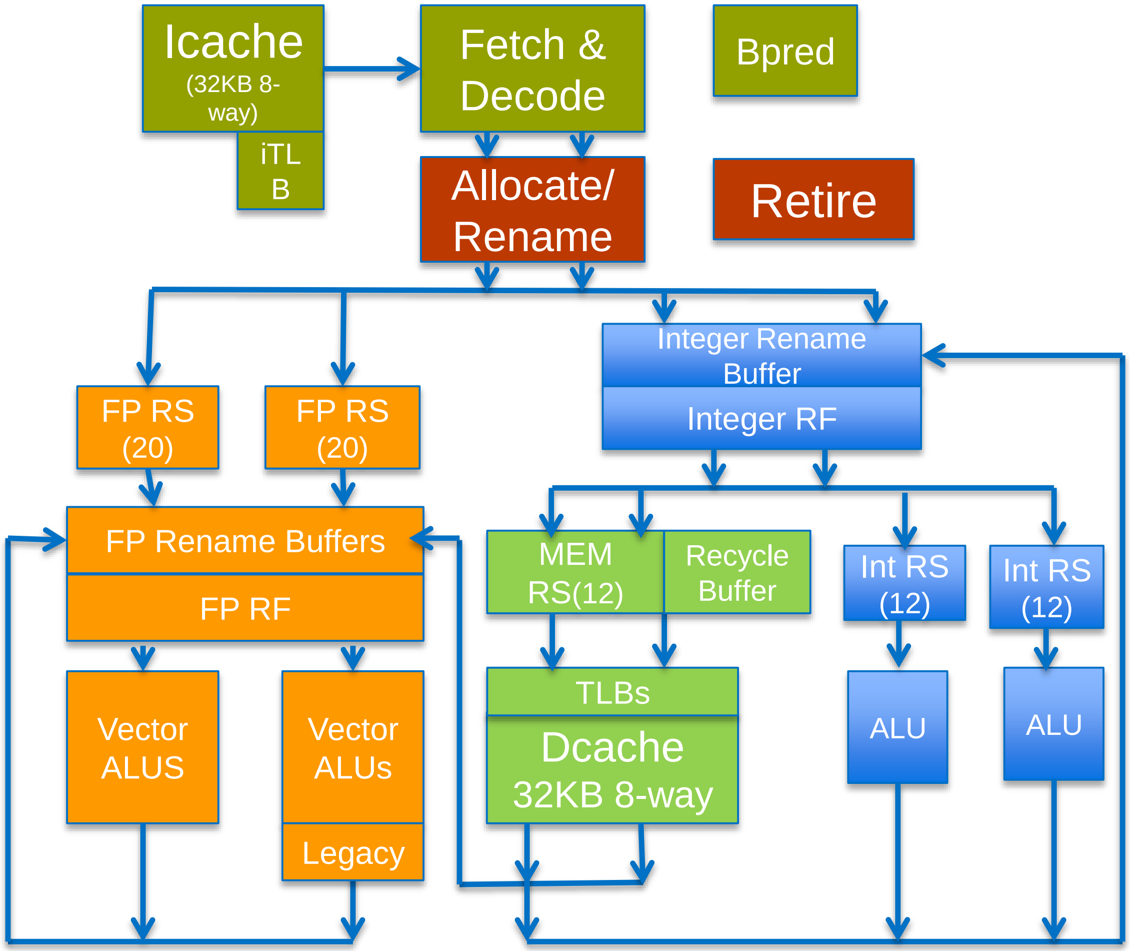
Intel Xeon Phi 72x0 Core & VPU

Balanced power efficiency, single thread performance and parallel performance

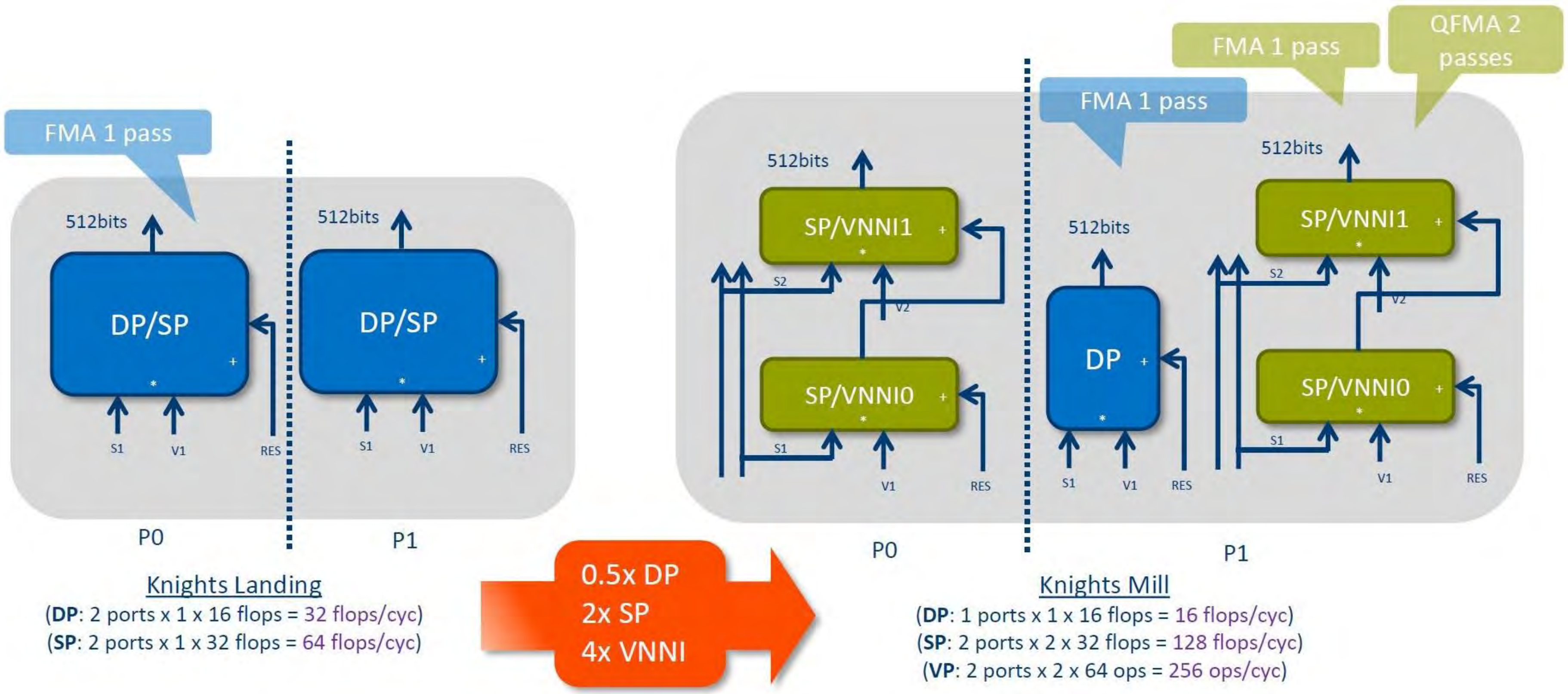
2-wide Out-of-order core

4 SMT threads

- 72 in-flight instructions.
- 6-wide execution
- 64 SP and 32 DP Flop/cycle
- Two-level TLB. Large page support
- Gather/Scatter engine
- Unaligned load/store
- Core resources **shared** or **dynamically repartitioned** between active threads
- General purpose IA core
- Cache BW:
  - 2 line read L1$, 1line write L1$
  - 0.5 line read L2$, 0.25 line write L2$



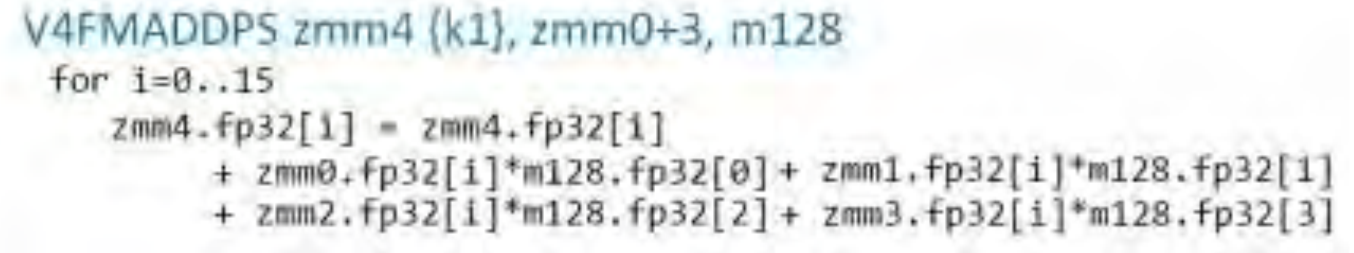
The Intel Xeon Phi 72x5 VPU (Knights Mill)



Knights Landing
(**DP**: 2 ports x 1 x 16 flops = 32 flops/cyc)
(**SP**: 2 ports x 1 x 32 flops = 64 flops/cyc)

Knights Mill
(**DP**: 1 ports x 1 x 16 flops = 16 flops/cyc)
(**SP**: 2 ports x 2 x 32 flops = 128 flops/cyc)
(**VP**: 2 ports x 2 x 64 ops = 256 ops/cyc)

Schematic of the QFMA instruction which implements a matrix vector multiplication, $M = 16; N = 1; K = 4$ with a latency of 13 cycles.



```
V4FMADDPS zmm4 {k1}, zmm0+3, m128
  for i=0..15
     zmm4.fp32[i] = zmm4.fp32[i]
              + zmm0.fp32[i]*m128.fp32[0] + zmm1.fp32[i]*m128.fp32[1]
              + zmm2.fp32[i]*m128.fp32[2] + zmm3.fp32[i]*m128.fp32[3]
```

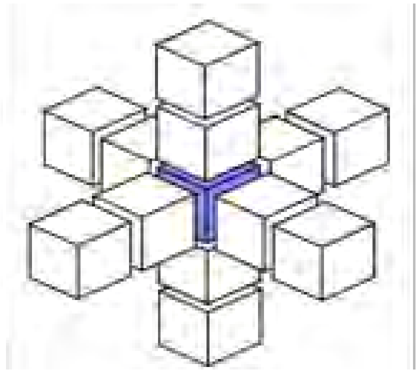
Two Representative Codes for Seismic WP

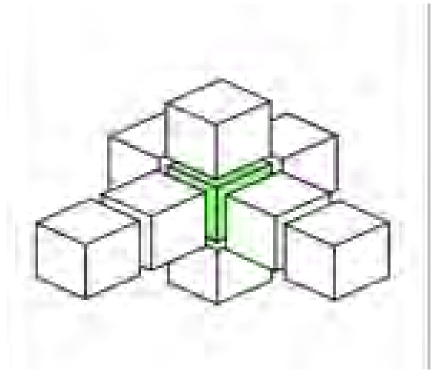
AWP-ODC-OS

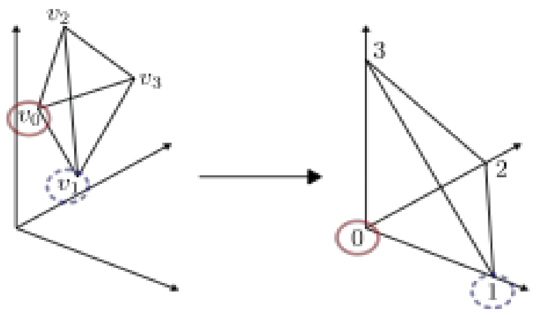
- Finite difference scheme: 4th order in space, 2nd order in time
- Staggered-grid, velocity/stress formulation of elastodynamic eqns with frequency dependent attenuation
- Memory bandwidth bound

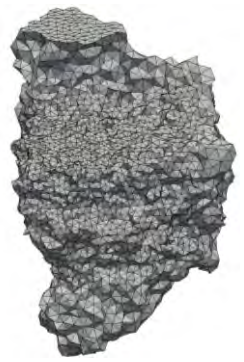
EDGE

- Discontinuous Galerkin Finite Element Method (DG-FEM)
- Unstructured tetrahedral meshes
- Small matrix kernels in inner loops
- Compute bound for higher orders



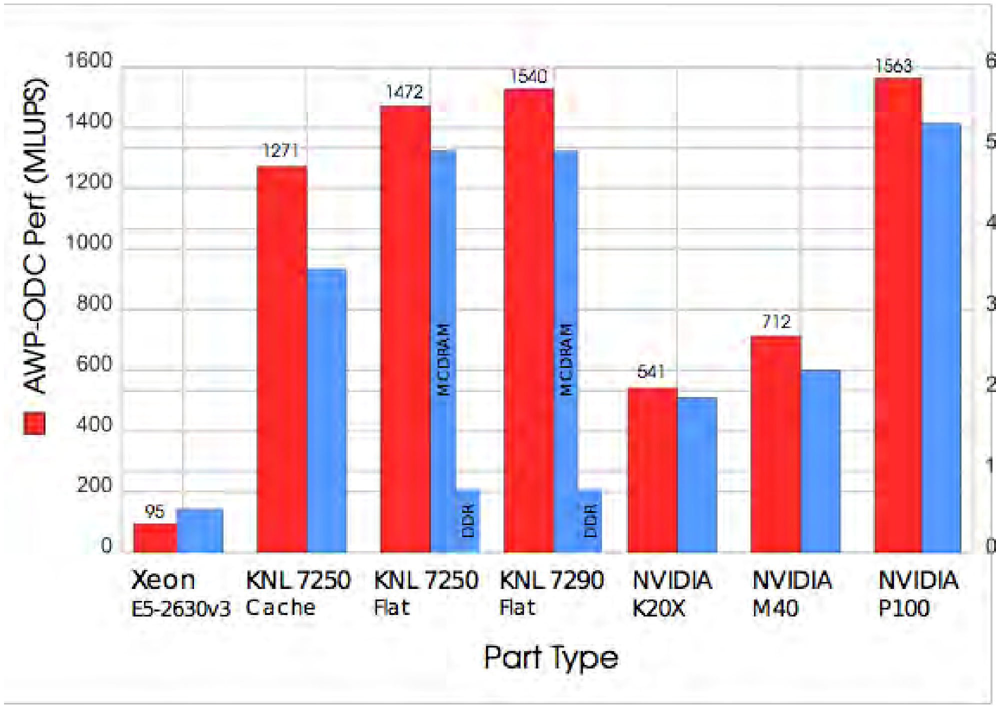






Architecture Comparison

- Xeon Phi KNL 7290:
  - 2x speedup over NVIDIA K20X; 97% of NVIDIA Tesla P100 performance
- Memory bandwidth
  - accurately predicts performance of architectures (as measured by STREAM and HPCG-SpMv)



Single node performance comparison of AWP-ODC-OS on a variety of architectures. Also displayed is the bandwidth of each architecture, as measured by a STREAM and HPCG-SpMv [ISC_17_2].

Algorithm: ADER DG-FEM

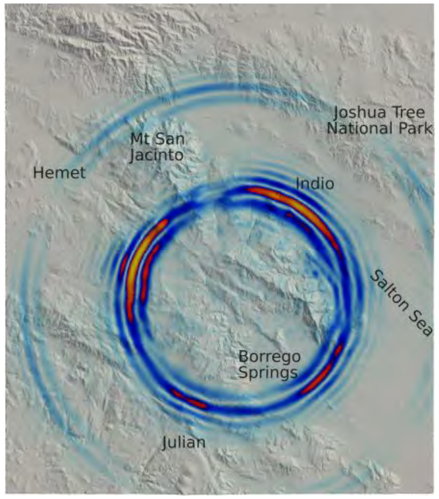
Connecting Two Worlds

Accurate earthquake simulations

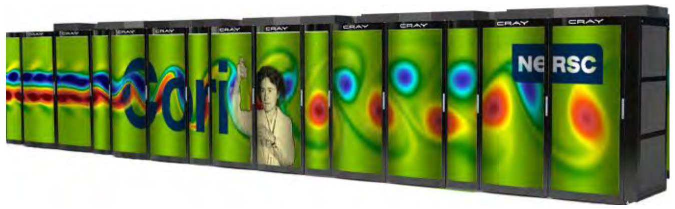
- Unstructured tet meshes for high geometric complexities
- High order DG-FEM for numerical efficiency, face-only stencils, and low dispersion errors
- Many elements for resolved, high frequencies

Big computers

- Vectorization: SIMD
- (Shared memory parallelization for manycore architectures)
- (Distributed memory parallelization for thousands of nodes)



Example of hypothetical seismic wave propagation with mountain topography using EDGE. Shown is the surface of the computational domain covering the San Jacinto fault zone between Anza and Borrego Springs in California. Colors denote the amplitude of the particle velocity, where warmer colors correspond to higher amplitudes.



Cori @ NERSC, image: http://www.nersc.gov/news-publications/nersc-news/nersc-center-news/2015/early-users-to-test-new-burst-buffer-on-cori/

Fully Discrete Form

Local Update

$$Q_k^{*,n_k+1} = Q_k^{n_k} + \mathcal{V}_k(\mathcal{T}_k) - \sum_{i=1}^{4} F_{k,i}^{-}(\mathcal{T}_k)$$

Neighboring Update

$$Q_k^{n_k+1} = Q_k^{*,n_k+1} - \sum_{i=1}^{4} \mathcal{F}_{k,i}^{+}(\mathcal{T}_{k_i})$$

Implemented as two functions (local, neighboring).
These drive the performance of the entire code.

Time

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

Volume

$$\mathcal{V}_k(\mathcal{T}_k) = \sum_{c=1}^{3} \hat{A}_k^{\xi_c} \boxed{\mathcal{T}_k} K^{\xi_c} M^{-1}$$

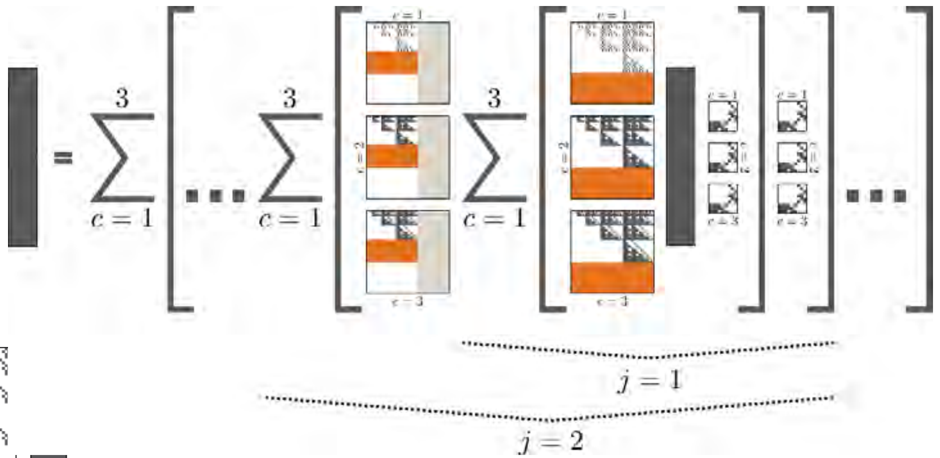
Flux

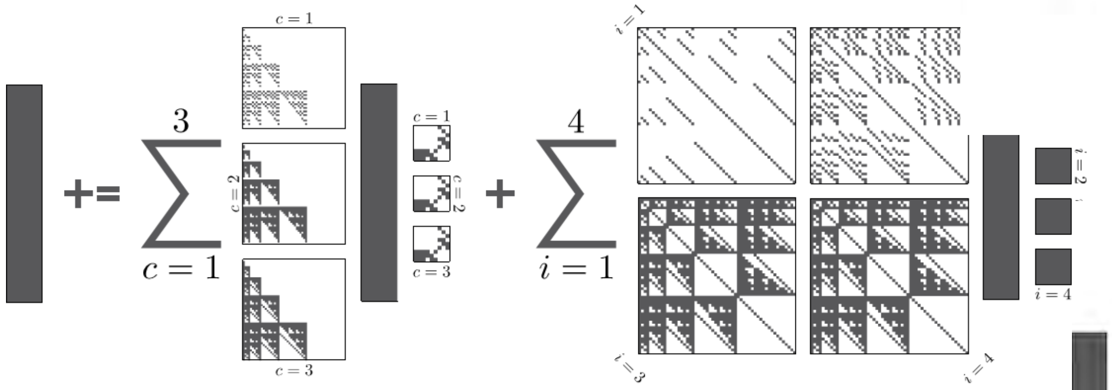
$$\mathcal{F}_{k,i}^{-}(\mathcal{T}_k) = \frac{|S_i|}{|J_k|} \hat{A}_{k,i}^{-} \boxed{\mathcal{T}_k} F^{-,i} M^{-1} \qquad \mathcal{F}_{k,i}^{+}(\mathcal{T}_{k_i}) = \frac{|S_i|}{|J_k|} \hat{A}_{k,i}^{+} \boxed{\mathcal{T}_{k_i}} F^{+,i,j,k} M^{-1}$$

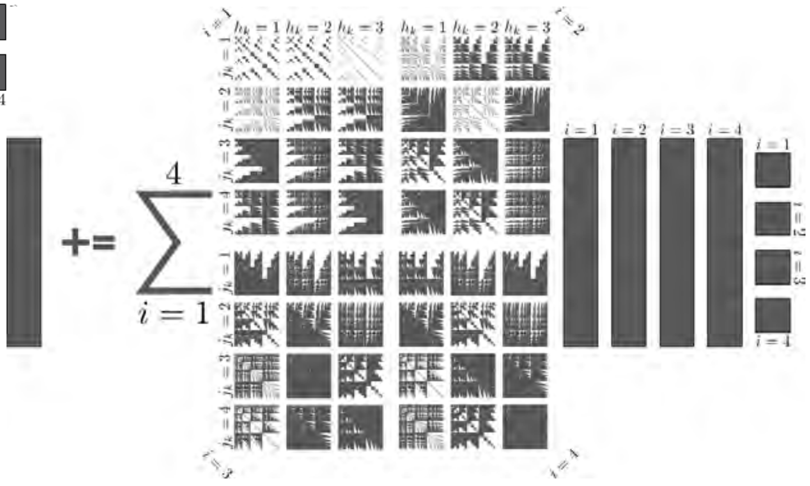
EDGE’s Compute Kernels

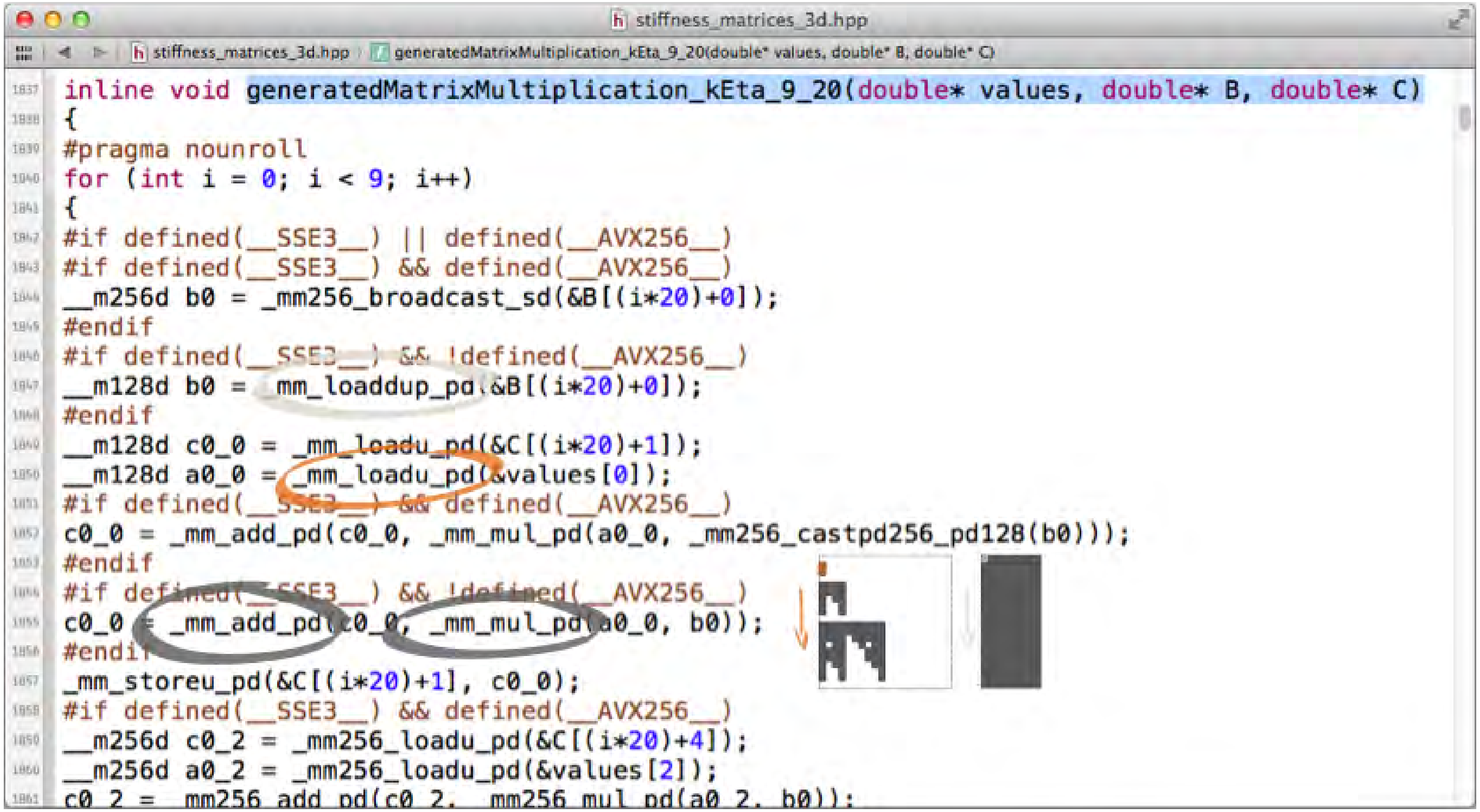
Small Matrix-Matrix multiplications, (convergence order 6): $9x9, 56x9, 56x35$

A priori known sparsity patterns








stiffness_matrices_3d.hpp

stiffness_matrices_3d.hpp › generatedMatrixMultiplication_kEta_9_20(double* values, double* B, double* C)

```
inline void generatedMatrixMultiplication_kEta_9_20(double* values, double* B, double* C)
{
#pragma nounroll
for (int i = 0; i < 9; i++)
{
#if defined(__SSE3__) || defined(__AVX256__)
#if defined(__SSE3__) && defined(__AVX256__)
__m256d b0 = _mm256_broadcast_sd(&B[(i*20)+0]);
#endif
#if defined(__SSE3__) && !defined(__AVX256__)
__m128d b0 = _mm_loaddup_pd(&B[(i*20)+0]);
#endif
__m128d c0_0 = _mm_loadu_pd(&C[(i*20)+1]);
__m128d a0_0 = _mm_loadu_pd(&values[0]);
#if defined(__SSE3__) && defined(__AVX256__)
c0_0 = _mm_add_pd(c0_0, _mm_mul_pd(a0_0, _mm256_castpd256_pd128(b0)));
#endif
#if defined(__SSE3__) && !defined(__AVX256__)
c0_0 = _mm_add_pd(c0_0, _mm_mul_pd(a0_0, b0));
#endif
_mm_storeu_pd(&C[(i*20)+1], c0_0);
#if defined(__SSE3__) && defined(__AVX256__)
__m256d c0_2 = _mm256_loadu_pd(&C[(i*20)+4]);
__m256d a0_2 = _mm256_loadu_pd(&values[2]);
c0_2 = _mm256_add_pd(c0_2, _mm256_mul_pd(a0_2, b0));
```

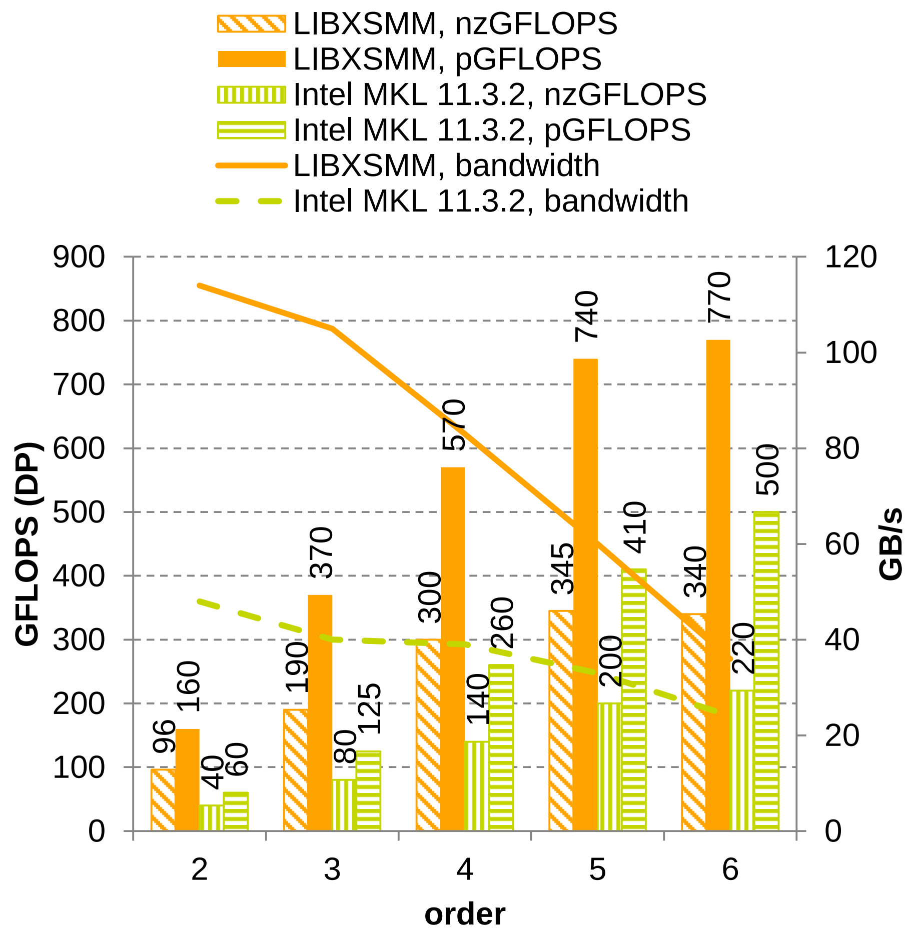
Illustrative example of offline code generation for an O4 (P3) elements stiffness matrix multiplication. Shown are intrinsics for a sparse matrix-matrix multiplication and the SSE3 vector instruction set.

Remark: EDGE uses assembly kernels, obtained through just-in-time code generation, and fully vectorized AVX-512 instructions for sparse kernels through fused simulations [ISC17].

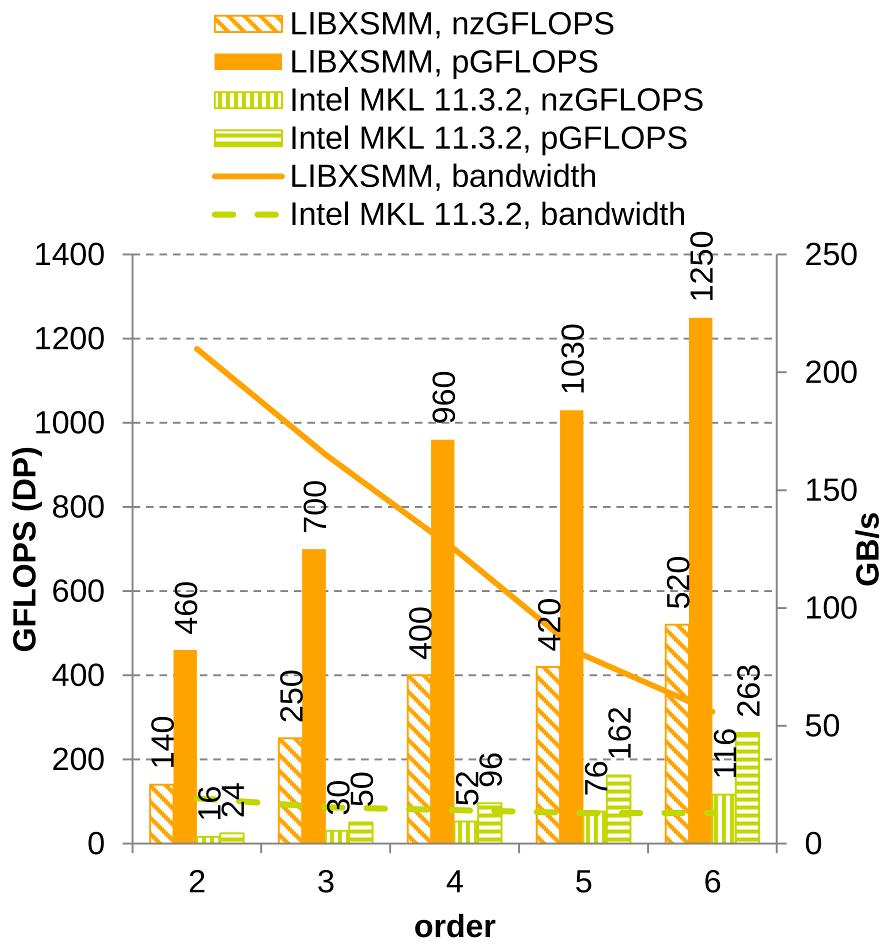
Our solution: JIT-based coding via LIBXSMM: https://github.com/hfp/libxsmm/

EDGE LOH.1 Benchmark Performance, all dense

Dual Socket Xeon E5-2697v4



Single Socket Xeon Phi 7250



Mixed / Lower Precision

What about FP32 vs. FP64 for Convergence?

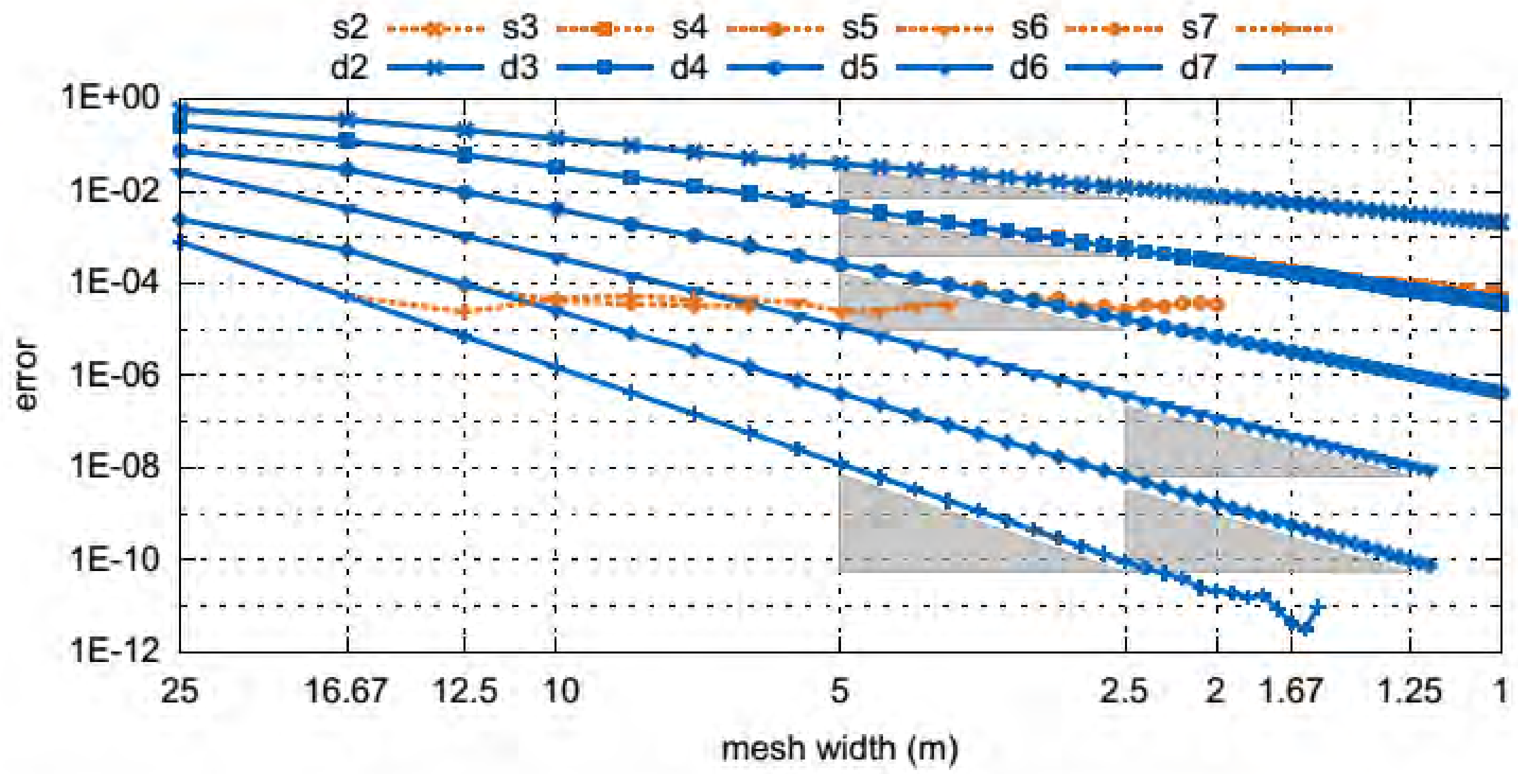
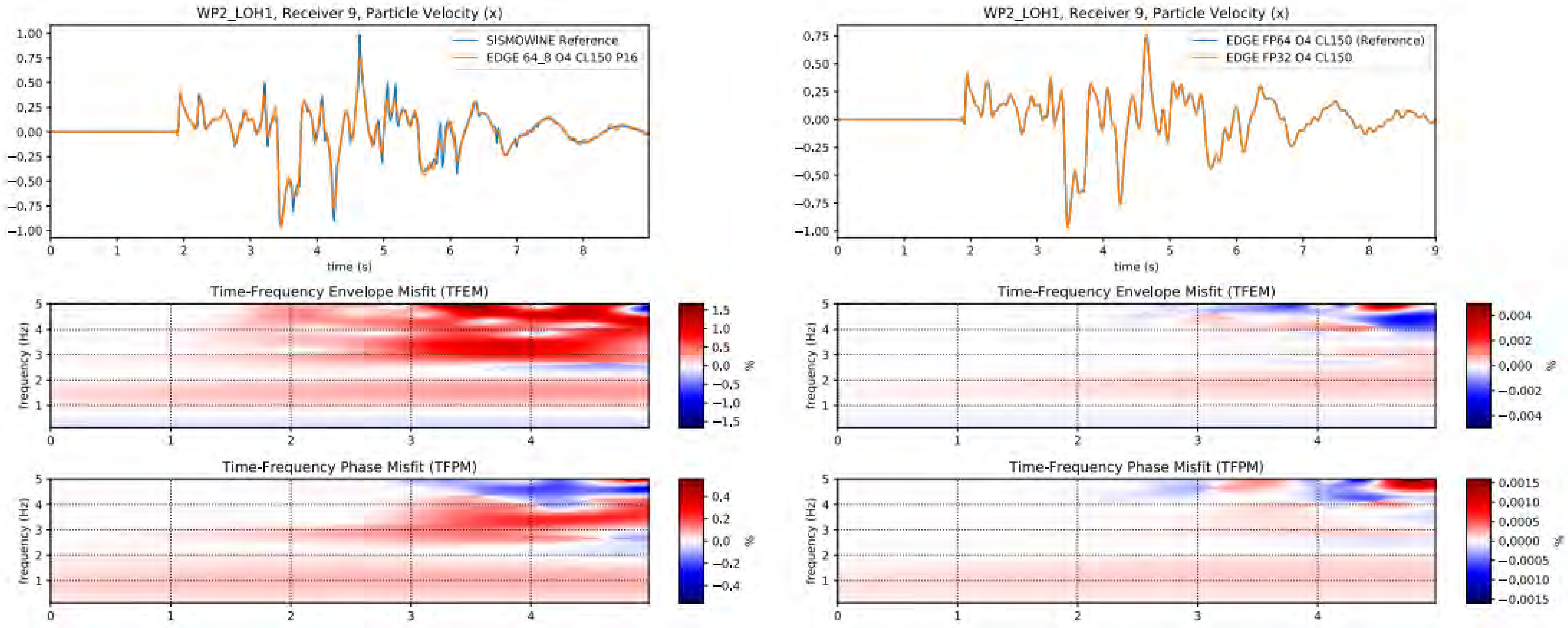


Fig. 2. Convergence of the synthetic benchmark for orders $\mathcal{O}2 - \mathcal{O}7$. Shown is the mesh width $100/N_{1D}$ against the error in the L^{∞}-norm for the variable σ_{yz}. Orange dashed lines represent single precision, blue solid lines double precision. The slopes of the gray triangles next to the respective curves illustrate the mathematical convergence rate.

FP32 vs. FP64 for the LOH.1 Benchmark



	location (m)			$\mathcal{O} = 4$		$\mathcal{O} = 5$		$\mathcal{O} = 6$	
	x	y	z	EM	PM	EM	PM	EM	PM
max				1.43	0.39	1.48	0.35	1.47	0.41
64-bit vs. 32-bit, max				10^{-2}	10^{-3}	10^{-2}	10^{-3}	10^{-2}	10^{-2}

Fused Simulations

(or how to leverage unbalanced SIMD platforms efficiently for scientific computing)

Fused Simulations

Exploits inter-simulation parallelism:

- Full vector operations, even for sparse matrix operators
- Automatic memory alignment
- Read-only data shared among all runs
- Lower sensitivity to latency (memory & network)

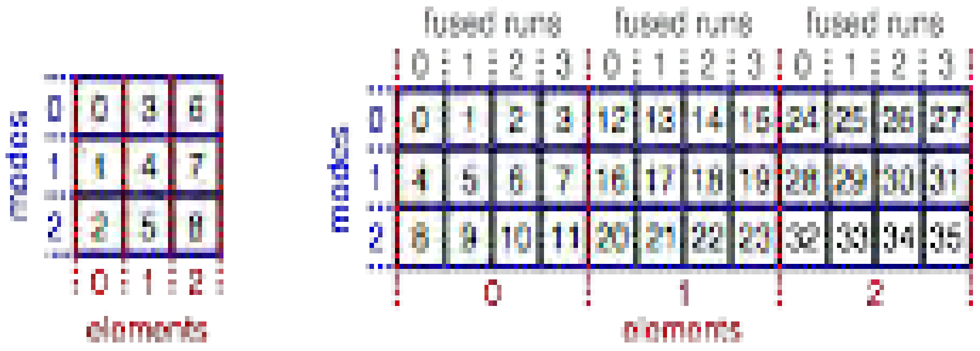


Illustration of the memory layout for fused simulations in EDGE. Shown is a third order configuration for line elements and the advection equation. Left: Single forward simulation, right: 4 fused simulations

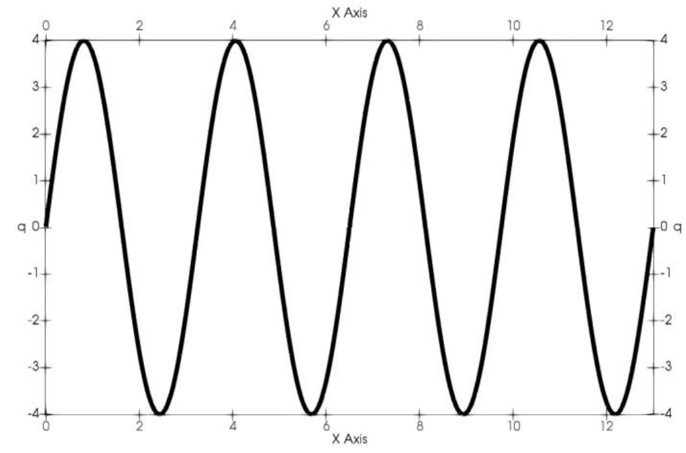
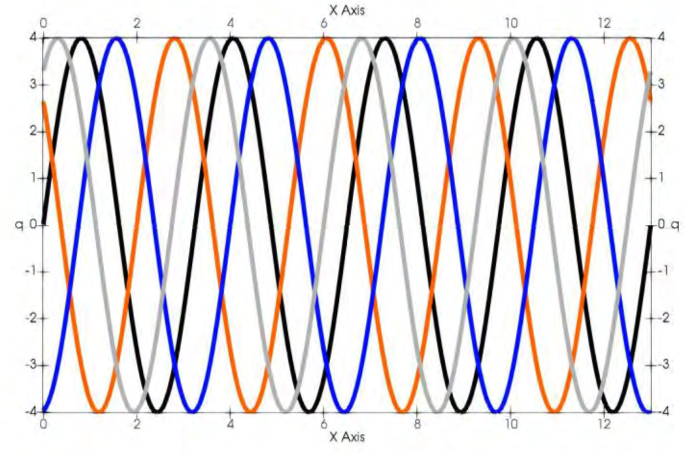



Illustration of fused simulations in EDGE for the advection equation using line elements. Top: Single forward simulation, bottom: 4 fused simulations.

Fused Simulations: Performance

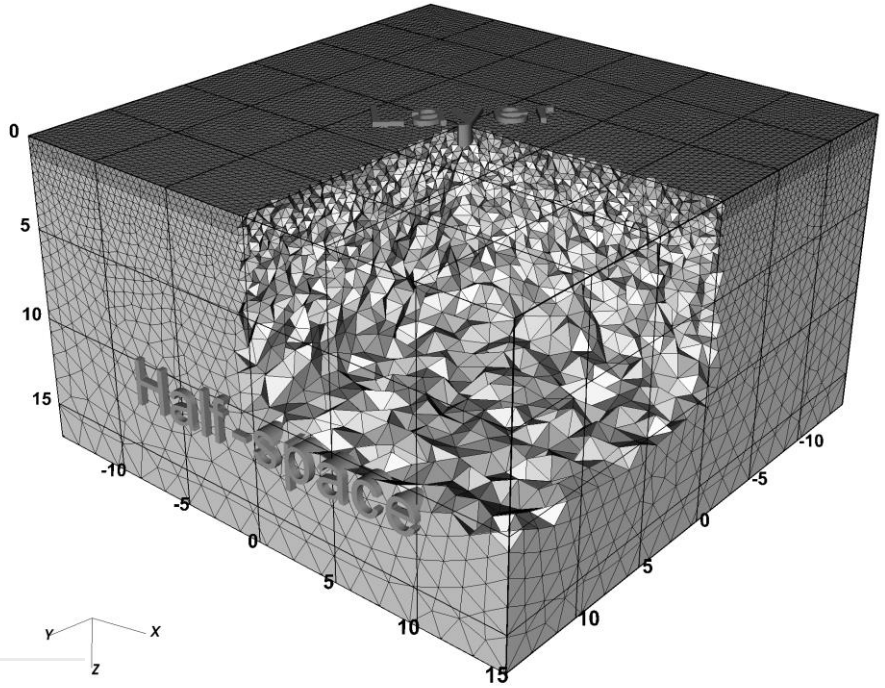
Orders: 2-6 (non-fused), 2-4 (fused)

Unstructured tetrahedral mesh: 350,264 elements

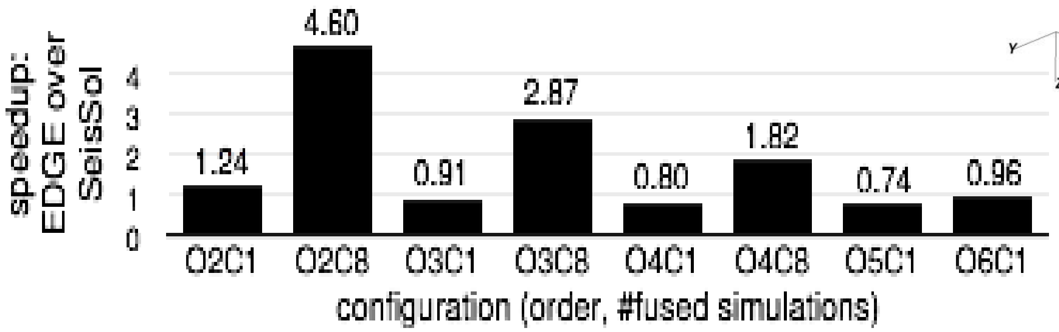
Single node of Cori-II (68 core Intel Xeon Phi x200, code-named Knights Landing)

EDGE vs. SeisSol (GTS, git-tag 201511)

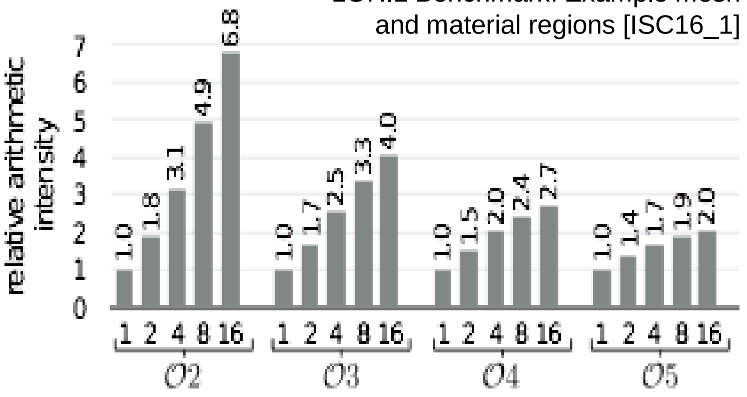
Speedup: 2-5x



LOH.1 Benchmark: Example mesh and material regions [ISC16_1]

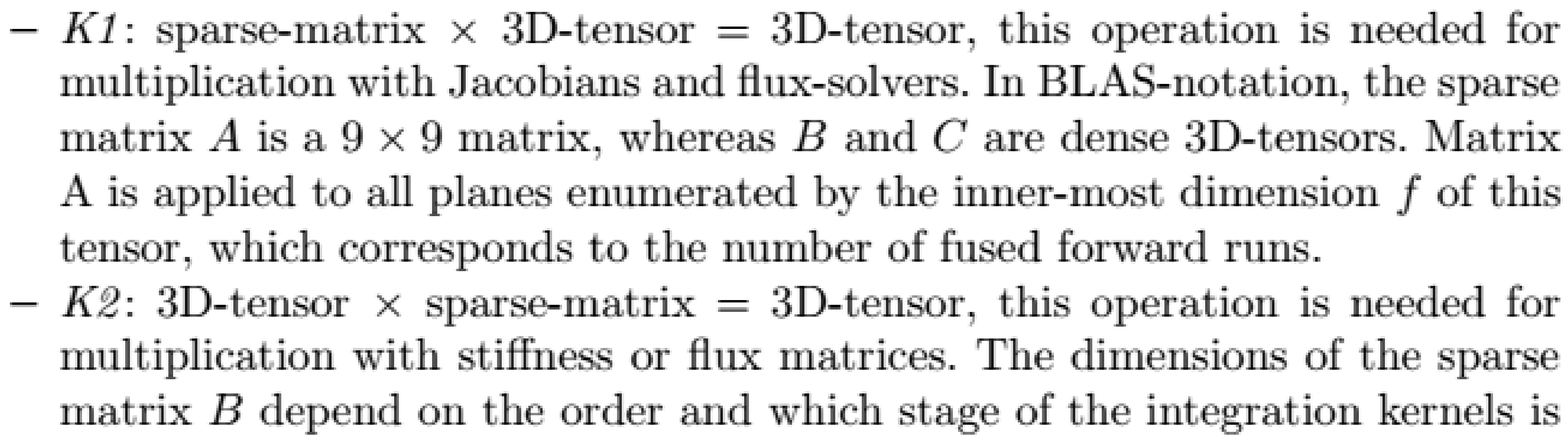


Speedup of EDGE over SeisSol (GTS, git-tag 201511). Convergence rates O2 − O6: single non-fused forward simulations (O2C1-O6C1). Additionally, per-simulation speedups for orders O2−O4 when using EDGE's full capabilities by fusing eight simulations (O2C8-O4C8). [ISC17_1]



SIMD

Code Generation Via LIBXSMM



- *K1*: sparse-matrix $\times$ 3D-tensor $=$ 3D-tensor, this operation is needed for multiplication with Jacobians and flux-solvers. In BLAS-notation, the sparse matrix A is a 9×9 matrix, whereas B and C are dense 3D-tensors. Matrix A is applied to all planes enumerated by the inner-most dimension f of this tensor, which corresponds to the number of fused forward runs.
- *K2*: 3D-tensor $\times$ sparse-matrix $=$ 3D-tensor, this operation is needed for multiplication with stiffness or flux matrices. The dimensions of the sparse matrix B depend on the order and which stage of the integration kernels is

K1 is trivially vectorize-able and block-able, we cannot leverage any matrix multiplication instruction without transposing the DOF tensor, but FLOPs are very low, so we don’t care that much.

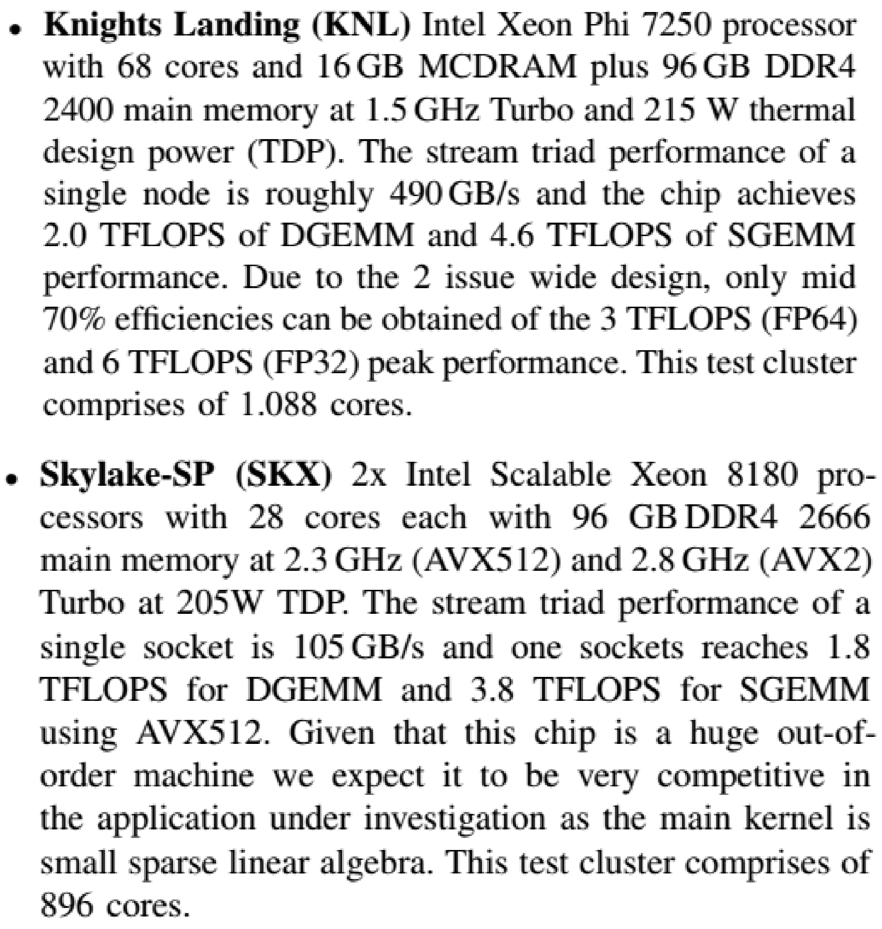
K2 Kernel Code Jitter

Algorithm 2 Code generator sketch of kernel $K2$, sparse matrix B is stored in CSC format.

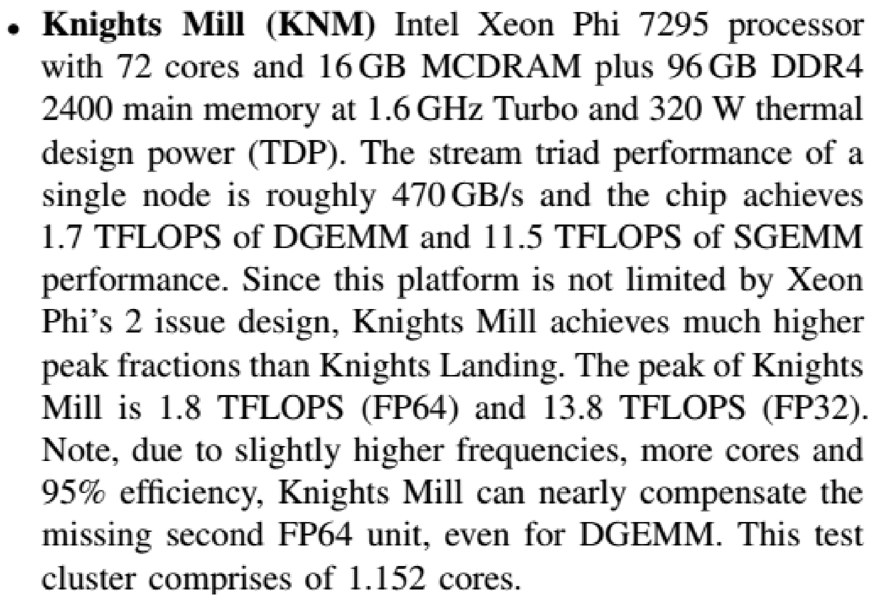
```
$nb \leftarrow \lceil \#\text{modes}/\text{scratchpad\_size} \rceil$
for all $m = 1$ to #quantities do
    for all $blk = 1$ to $nb$ do
        $n_0 \leftarrow (blk - 1) \cdot \text{scratchpad\_size}$
        for all $n = 1$ to scratchpad_size do $c_n[1:f] \leftarrow C[m][n_0+n][1:f]$ end for
        for all $k = 1$ to #modes do
            for all $n = 1$ to scratchpad_size do
                $b_{\#\text{Entries}} \leftarrow \text{col}_B[n_0+n+1] - \text{col}_B[n_0+n]$
                for $l = 1$ to $b_{\#\text{Entries}}$ do
                    if $\text{row}_B[\text{col}_B[n_0+n]+l] == k$ then
                        $b[1:f] \leftarrow \text{broadcast}(B[\text{col}_B[n_0+n]+l])$
                        $c_n[1:f] \leftarrow \text{FMA}(A[m][k][1:f], b[1:f], c_n[1:f])$
                    end if
                end for
            end for
            for all $n = 1$ to $bksz_n$ do $C[m][n_0+n][1:f] \leftarrow c_n[1:f]$ end for
        end for
    end for
end for
```

if $\text{row}_B[\text{col}_B[n_0+n]+l] == k$, $\text{row}_B[\text{col}_B[n_0+n]+l+1] == k+1$, $\text{row}_B[\text{col}_B[n_0+n]+l+2] == k+2$ and finally $\text{row}_B[\text{col}_B[n_0+n]+l+3] == k+3$ a QFMA instruction can be issued.

Platforms Tested

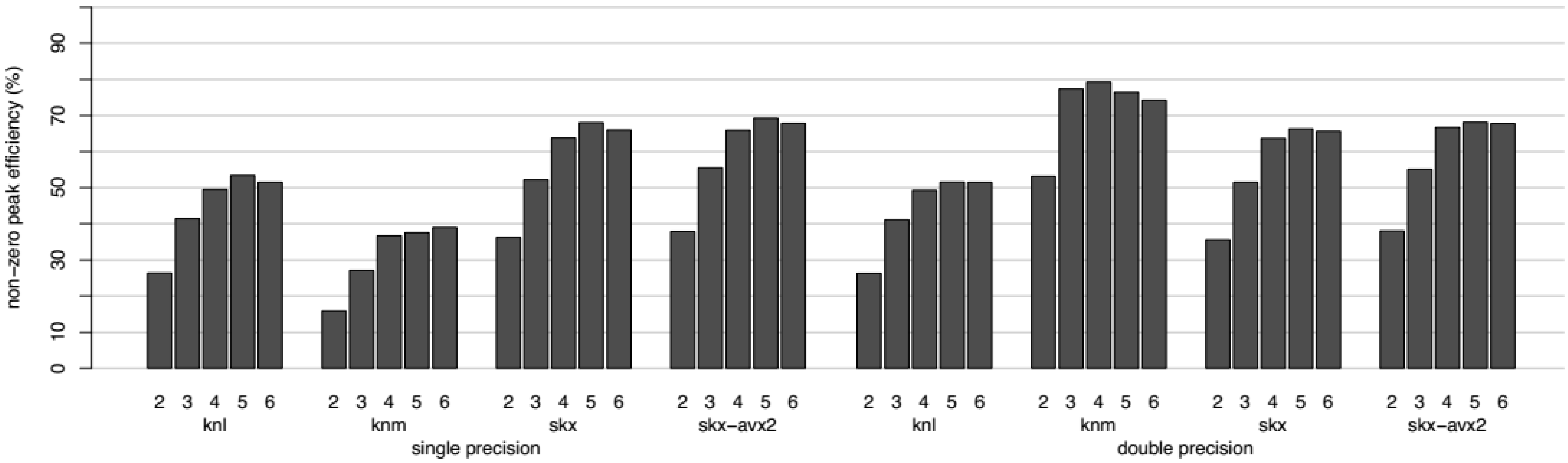


- **Knights Landing (KNL)** Intel Xeon Phi 7250 processor with 68 cores and 16 GB MCDRAM plus 96 GB DDR4 2400 main memory at 1.5 GHz Turbo and 215 W thermal design power (TDP). The stream triad performance of a single node is roughly 490 GB/s and the chip achieves 2.0 TFLOPS of DGEMM and 4.6 TFLOPS of SGEMM performance. Due to the 2 issue wide design, only mid 70% efficiencies can be obtained of the 3 TFLOPS (FP64) and 6 TFLOPS (FP32) peak performance. This test cluster comprises of 1.088 cores.
- **Skylake-SP (SKX)** 2x Intel Scalable Xeon 8180 processors with 28 cores each with 96 GB DDR4 2666 main memory at 2.3 GHz (AVX512) and 2.8 GHz (AVX2) Turbo at 205W TDP. The stream triad performance of a single socket is 105 GB/s and one sockets reaches 1.8 TFLOPS for DGEMM and 3.8 TFLOPS for SGEMM using AVX512. Given that this chip is a huge out-of-order machine we expect it to be very competitive in the application under investigation as the main kernel is small sparse linear algebra. This test cluster comprises of 896 cores.



- **Knights Mill (KNM)** Intel Xeon Phi 7295 processor with 72 cores and 16 GB MCDRAM plus 96 GB DDR4 2400 main memory at 1.6 GHz Turbo and 320 W thermal design power (TDP). The stream triad performance of a single node is roughly 470 GB/s and the chip achieves 1.7 TFLOPS of DGEMM and 11.5 TFLOPS of SGEMM performance. Since this platform is not limited by Xeon Phi's 2 issue design, Knights Mill achieves much higher peak fractions than Knights Landing. The peak of Knights Mill is 1.8 TFLOPS (FP64) and 13.8 TFLOPS (FP32). Note, due to slightly higher frequencies, more cores and 95% efficiency, Knights Mill can nearly compensate the missing second FP64 unit, even for DGEMM. This test cluster comprises of 1.152 cores.

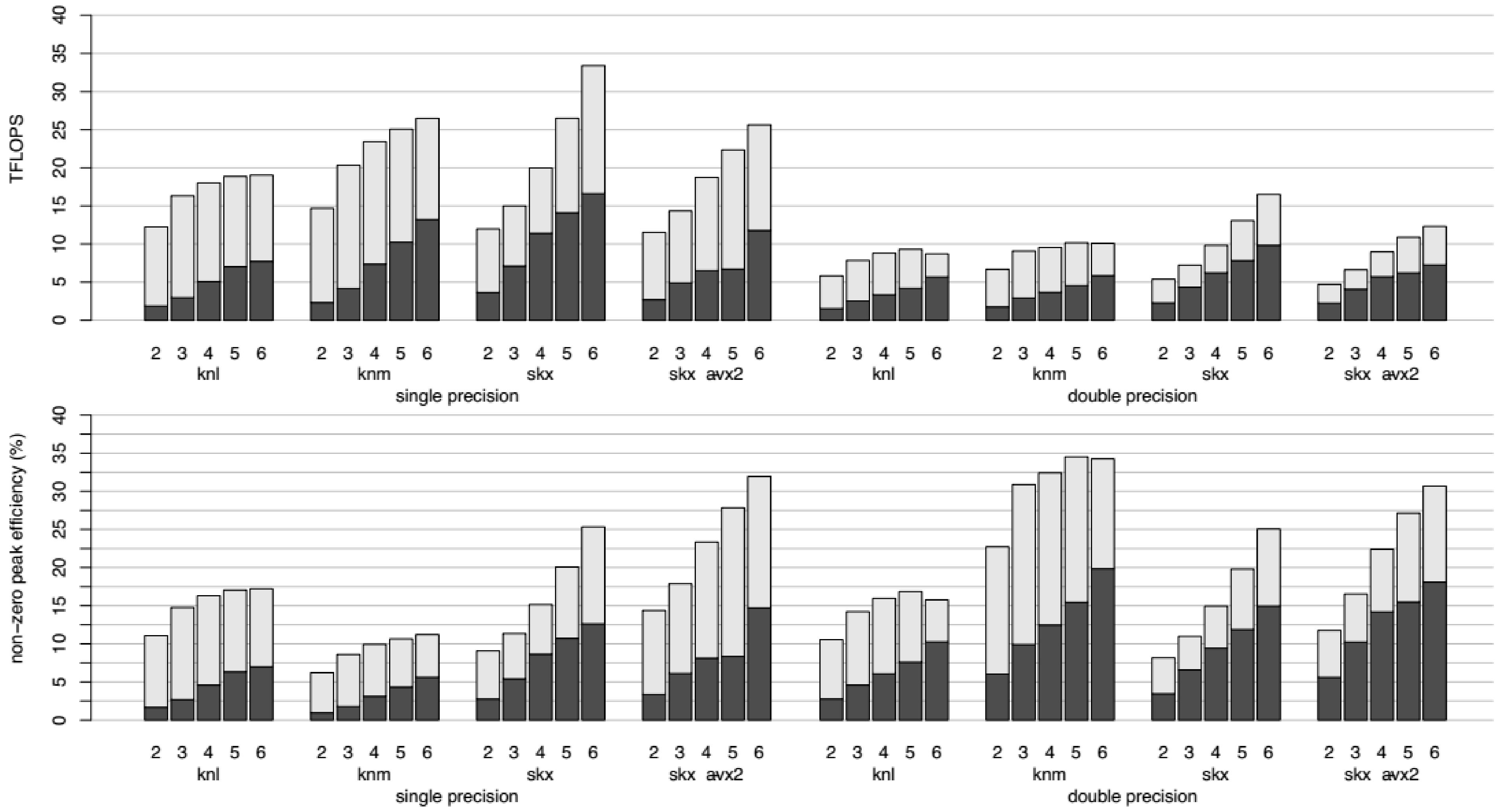
Kernel Performance out of hot L1$



FLOP-weighted average over all EDGE kernels

- KNL achieves 50% peak, (recall big GEMM is at 78% peak), here we do sparse*dense GEMM, overheads from exposed latencies (number of independent FMAs low)
- KNM is at up to 40% for FP32 and up to 80% for FP64
- SKX is at up to 70% peak

Performance for the LOH.1 benchmark in 16 nodes



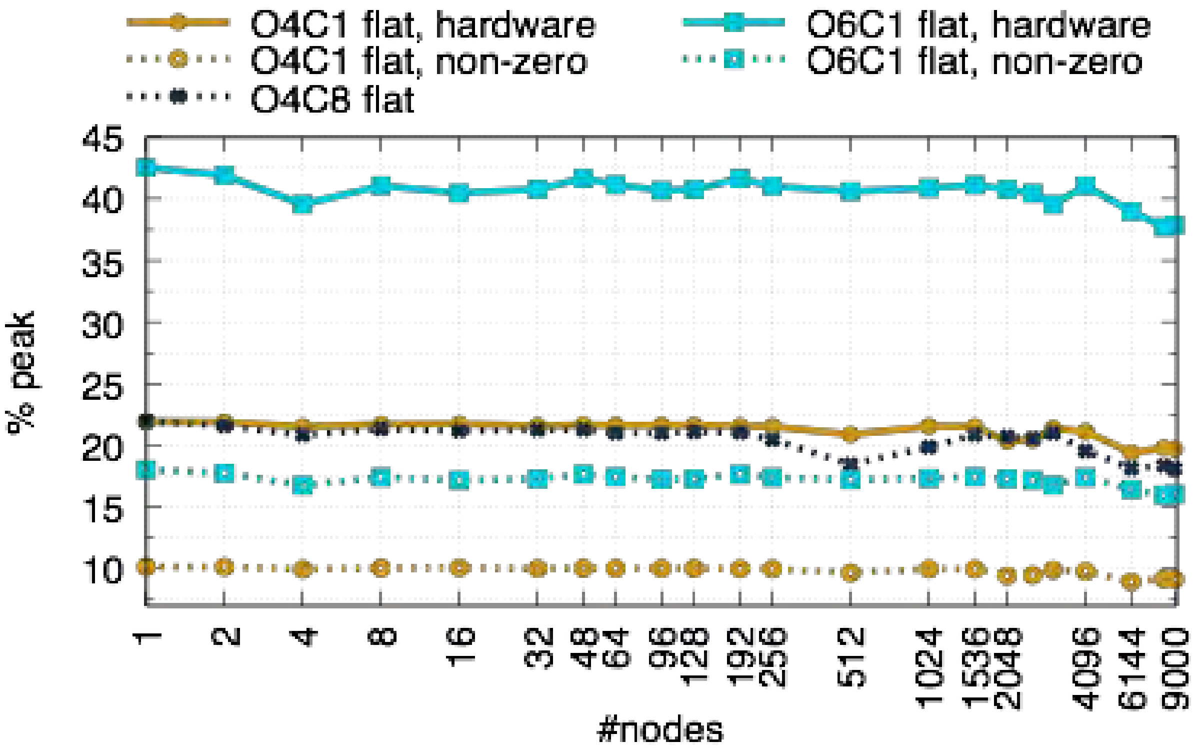
Dark grey: non-fused simulation

Light grey: fused simulation

Scaling

Reaching 10+ PFLOPS (FP64)

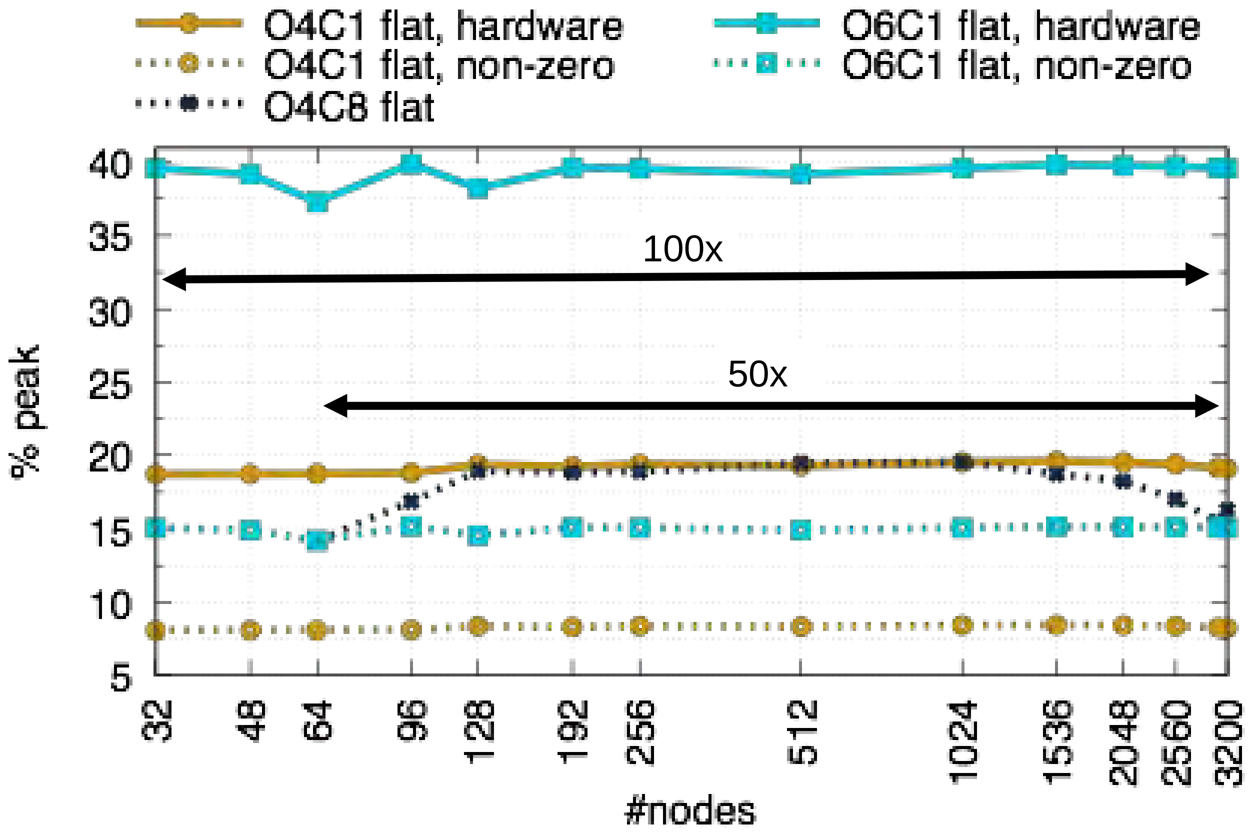
- Regular cubic mesh, 5 Tets per Cube, 4th order (O4) and 6th order (O6)
- Imitates convergence benchmark
- 276K elements per node
- 1-9000 nodes of Cori-II (9000 nodes = 612,000 cores)
- O6C1 @ 9K nodes: 10.4 PFLOPS (38% of peak)
- O4C8: @ 9K nodes: 5.0 PFLOPS (18% of peak)
- O4C8 vs. O4C1 @ 9K nodes: 2.0x speedup



Weak scaling study on Cori-II. Shown are hardware and non-zero peak efficiencies in flat mode. O denotes the order and C the number of fused simulations [ISC17_1].

Strong at the Limit: 50x and 100x

- Unstructured tetrahedral mesh: 172,386,915 elements
- 32-3200 nodes of Theta (64 core Intel Xeon Phi x200, code-named Knights Landing)
- 3200 nodes = 204,800 cores
- O6C1 @ 3.2K nodes: 3.4 PFLOPS (40% of peak)
- O4C8 vs. O4C1 @ 3.2K nodes: 2.0x speedup



Strong scaling study on Theta. Shown are hardware and non-zero peak efficiencies in flat mode. O denotes the order and C the number of fused simulations [ISC17_1].

Conclusions

- Both, FDM and DG-FEM, can be efficiently mapped to wide-SIMD architectures
  - Keep memory movement in mind
  - We need hardware aware formulation of kernels
  - Expect no magic from compiler ☺
- Lower precision needs to play an important role in these days as it sees enormous increase due to deep learning!
- DG-FEM (and SEM) can leverage architectural features which are intended to boost Deep Learning
- Fused simulations are new direction to further keep up with increasing flop/byte ratios

References

[ISC17_1] A. Breuer, A. Heinecke, Y. Cui: EDGE: Extreme Scale Fused Seismic Simulations with the Discontinuous Galerkin Method.
Proceedings of International Super Computing (ISC) High Performance 2017

[ISC17_2] J. Tobin, A. Breuer, C. Yount, A. Heinecke, Y. Cui: Accelerating Seismic Simulations Using the Intel Xeon Phi Knights Landing Processor
Proceedings of International Super Computing (ISC) High Performance 2017

[ISC16_1] A. Heinecke, A. Breuer, M. Bader: High Order Seismic Simulations on the Intel Xeon Phi Processor (Knights Landing).
High Performance Computing: 31st International Conference, ISC High Performance 2016, Frankfurt, Germany, June 19-23, 2016, Proceedings. http://dx.doi.org/10.1007/978-3-319-41321-1_18

[ISC16_2] A. Heinecke, A. Breuer, M. Bader: Chapter 21 - High Performance Earthquake Simulations.
In Intel Xeon Phi Processor High Performance Programming Knights Landing Edition.

[IPDPS16] A. Breuer, A. Heinecke, M. Bader: Petascale Local Time Stepping for the ADER-DG Finite Element Method.
In Parallel and Distributed Processing Symposium, 2016 IEEE International. http://dx.doi.org/10.1109/IPDPS.2016.109

[ISC15] A. Breuer, A. Heinecke, L. Rannabauer, M. Bader: High-Order ADER-DG Minimizes Energy- and Time-to-Solution of SeisSol.
In 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015.

[SC14] A. Heinecke, A. Breuer, S. Rettenberger, M. Bader, A.-A. Gabriel, C. Pelties, A. Bode, W. Barth, X.-K. Liao, K. Vaidyanathan, M. Smelyanskiy and P. Dubey: Petascale High Order Dynamic Rupture Earthquake Simulations on Heterogeneous Supercomputers.
In Supercomputing 2014, The International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, New Orleans, LA, USA, November 2014. Gordon Bell Finalist.

[ISC14] A. Breuer, A. Heinecke, S. Rettenberger, M. Bader, A.-A. Gabriel and C. Pelties: Sustained Petascale Performance of Seismic Simulations with SeisSol on SuperMUC.
In J.M. Kunkel, T. T. Ludwig and H.W. Meuer (ed.), Supercomputing — 29th International Conference, ISC 2014, Volume 8488 of Lecture Notes in Computer Science. Springer, Heidelberg, June 2014. 2014 PRACE ISC Award.

[PARCO13] A. Breuer, A. Heinecke, M. Bader and C. Pelties: Accelerating SeisSol by Generating Vectorized Code for Sparse Matrix Operators.
In Parallel Computing — Accelerating Computational Science and Engineering (CSE), Volume 25 of Advances in Parallel Computing. IOS Press, April 2014.

intel®
experience
what's inside™