EFFICIENCY OF HIGH ORDER SPECTRAL ELEMENT METHODS ON PETASCALE ARCHITECTURES

Maxwell Hutchinson, ***Alexander Heinecke***, Hans Pabst, Greg Henry, Matteo Parsani, and David Keyes

Parallel Computing Lab, Intel Labs, USA
2016-06-22 International Supercomputing Conference (ISC)
Frankfurt, Germany

Legal Disclaimer & Optimization Notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Rayleigh-Taylor Instability (RTI)

Occurs when a dense fluid is supported by a lighter one

- The dense fluid falls through the lighter fluid in “spikes”
- The light fluid rises through the dense fluid in “bubbles”

Pumps energy into the system at **small** scales

- Particularly difficult to model

Rayleigh-Taylor Instability Applications

- Natural convection in the oceans and atmosphere
- Heat transport in coolant in nuclear reactors
- Stellar remnants in supernova
- Fuel-ablator mix in inertial confinement fusion
  - RTI transports carbon-laden ablator into the hydrogen fuel
  - Carbon radiates energy away, lowering temperature
  - Hydrogen fails to fuse (“ignite”)

The RTI is a *priority research direction* for the national ignition facility of the USA at LLNL

Boussinesq single-mode RTI

Assume the fluids have nearly the same density

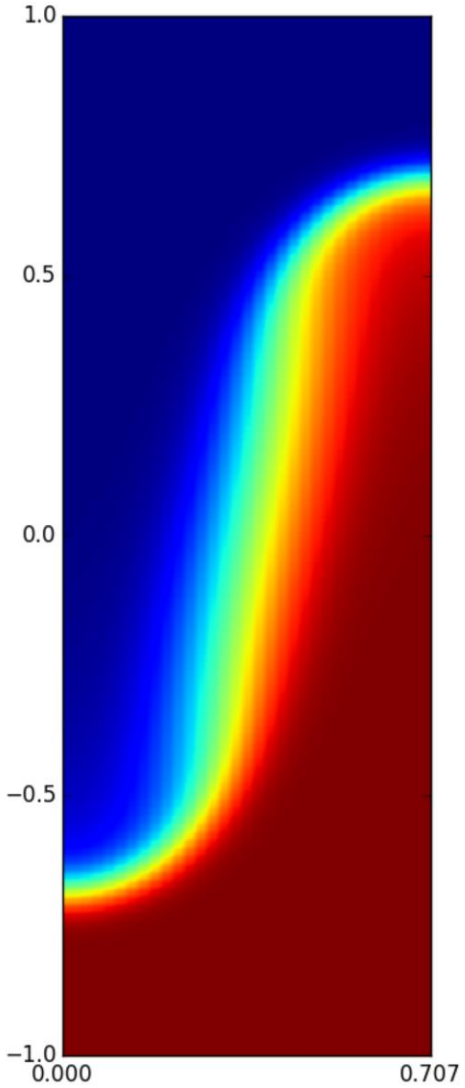
Assume the initial perturbation is small and sharp

Assume the fluids have the same viscosity

Restrict the initial condition to a single frequency

This makes the problem well-posed with two parameters:

- The Grashof number (Gr) compares the buoyant to viscous forces
- The Schmidt number (Sc) compares the diffusion of momentum and mass



Boussinesq single-mode RTI and model fitting

Now the problem is a mapping from the Grashof and Schmidt number to the trajectory of the rising bubble:

$$(Gr, Sc) \Rightarrow H(t)$$

Typically, mapping is evaluated via simulation of Navier-Stokes

- 3D non-linear PDEs are hard to solve correctly
- Direct numerical simulations are expensive: millions of core-hours typically

Instead, we can try to approximate the dynamics with simpler buoyancy-drag models

- ODEs are much easier to solve numerically

$$\ddot{H} = \frac{C_0 A g \lambda^2 H + C_1 \lambda^2 \dot{H}^2 + C_2 \lambda H \dot{H}}{C_3 \lambda^3 + C_4 \lambda^2 H}$$

The ODE approximate has undetermined parameters: the C's

- *Let's fit them! -> Our approach*
- "Nonlinear supervised learning"

Generating the data set

We want to cover the two-dimensional (Gr, Sc) input space

We are interested specifically in the error in the trajectory H(t)

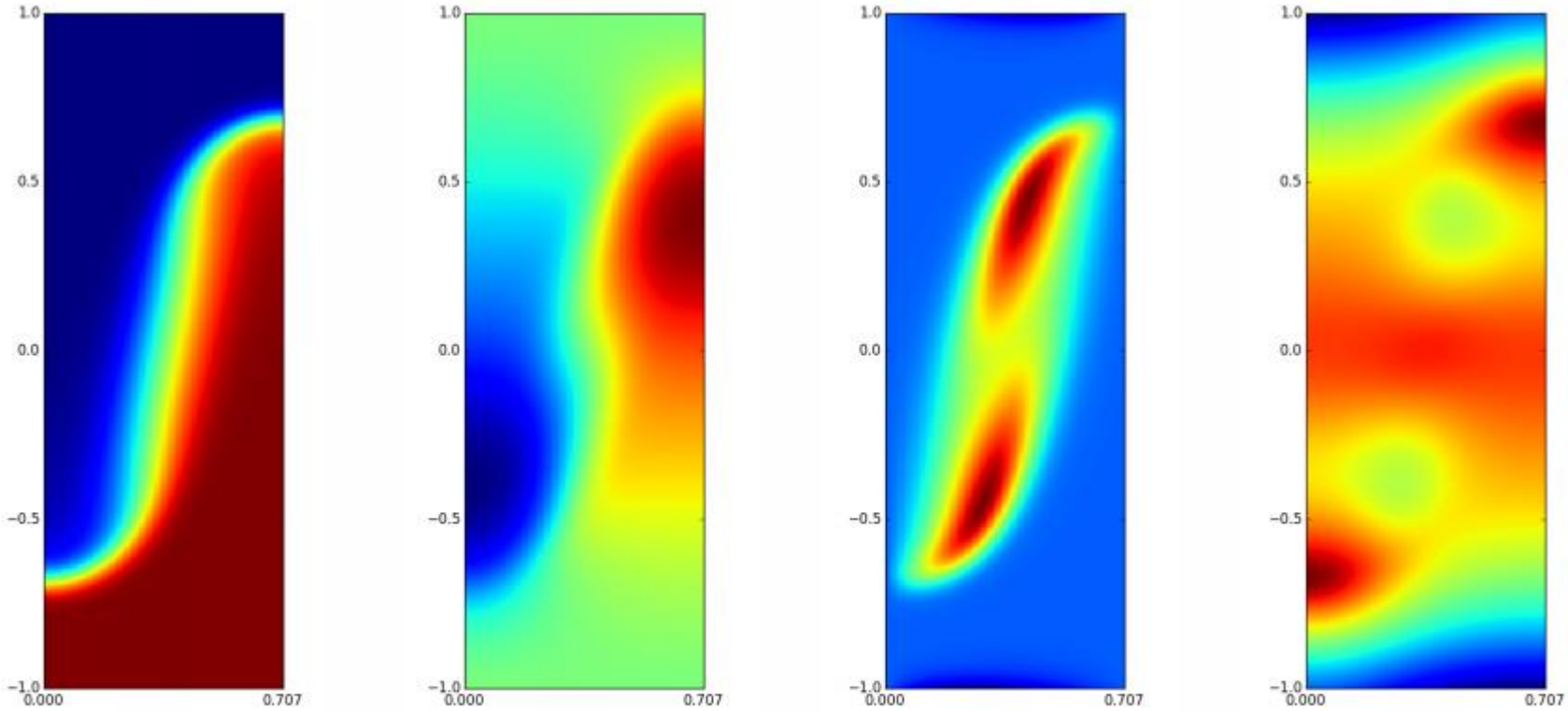
We have an error scale based on the quality of the ODE

- If the training data is much more accurate than the ODE, it is overkill

What is the most efficient discretization for generating this data?

- Which discretization that satisfies our accuracy requirement consumes the fewest cycles?

Several Variables at the End of the Simulation



(a) Scalar (b) Vertical velocity (c) Vorticity (d) Pressure

NekBox

NekBox is a stripped-down of Nek5000 for box-shaped domains

NekBox solves the incompressible Navier-Stokes equations:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{1}{\rho}\nabla p + \nu \nabla^2 u + f \qquad \nabla \cdot u = 0$$

Incl. advection-diffusion equations for scalar variables such as temperature or mass fractions.

Nek uses the spectral element method (SEM):

- A tensor product of Gauss-Lobatto-Legendre (GLL) quadrature points within each element -> N^3 DOFs per element
- Continuity across elements -> forming a mesh (using direct stiffness summation)

Operators are written as element local operators:

$$A = (A_x \times I_y \times I_z) + (I_x \times A_y \times I_z) + (I_x \times I_y \times A_z)$$

which reduces the complexity from O(N^6) to O(N^4).

Spectral Element Method (SEM)

Spatial discretization with two levels:

- Unstructured mesh of elements coupled by continuity at faces, edges
  - ***only nearest neighbours communication with depth 1, independent*** of the polynomial order
- Tensor product of Gauss-Lobatto-Legendre points within each element
  - ***small dense matrix multiplication*** (DGEMM) is the key kernel with Inner dimension (k) being the 1D size, not the 3D size, e.g. 4, 8, 16, 32

key ingredients for scalable and efficient code!

The mesh is good at matching geometry and boundary conditions

- Polynomial convergence w.r.t. mesh refinement

The GLL points are highly accurate

- Exponential convergence w.r.t. polynomial order

NekBox's main compute routines

A typical NekBox run spends <1% in sparse computations & communications, ~40% in vector-vector or matrix-vector operations**, ~60%** matrix-matrix operations.

Helmholtz operator:

```
Hu(:,:,:) = gx(:,:,:) * matmul(Kx(:,:), reshape(u, (/N, N*N/)))
do i = 1, n
  Hu(:,:,i) += gy(:,:,i) * matmul(u(:,:,i), KyT(:,:))
enddo
Hu(:,:,:) += gz(:,:,:) * matmul(reshape(u, (/N*N, N/)), KzT(:,:))
Hu(:,:,:) = h1 * Hu(:,:,:) + h2 * M(:,:,:) * u(:,:,:)
```

Basis transformation:

```
tmp_x = matmul(Ax, u)
do i = 1, n
  tmp_y(:,:,i) = matmul(tmp_x(:,:,i), AyT)
enddo
v = matmul(tmp_y, AzT)
```

Gradient calculation:

```
dudx = matmul(Dx, u)
do i = 1, n
  dudy(:,:,i) = matmul(u(:,:,i), DyT)
enddo
dudz = matmul(u, DzT)
```

→ **Batched GEMM is not beneficial as it would mean loosing locality**

LIBXSMM: Implementation

Interface (C/C++ and FORTRAN API)

Simplified interface for matrix-matrix multiplications

- $c_{m \times n} = c_{m \times n} + a_{m \times k} * b_{k \times n}$ (also full xGEMM)

Highly optimized assembly code generation (inline, *.s, JIT byte-code)

- SSE3, AVX, AVX2, IMCI, and AVX-512
- AVX-512 code quality
  - Maximizes number of immediate operands
  - Limits instruction width to 16 Byte/cycle

High level code optimizations

- Implicitly aligned leading dimension (LDC) – allows aligned store instr.
- Aligned load instructions
- Sophisticated data prefetch

License

- Open Source Software (BSD 3-clause license)*, https://github.com/hfp/libxsmm

Element Updates with non-temporal Stores

- NekBox performs vector stores that overwrite memory on necessarily unaligned data
- Regular unaligned stores would issue RFOs (read for ownership), which consume read BW
- Aligned non-temporal stores via peeling allow us to override the data without RFOs, increasing “useful” bandwidth

```
void stream_vector_copy( const double* i_a,
                         double*       io_c,
                         const int     i_length) {
  int l_n = 0;
  int l_trip_prolog = 0;
  int l_trip_stream = 0;

  /* init the trip counts to determine aligned middle section */
  stream_init( i_length, (size_t)io_c, &l_trip_prolog, &l_trip_stream );

  /* run the prologue */
  for ( ; l_n < l_trip_prolog;  l_n++ ) {
    io_c[l_n] = i_a[l_n];
  }
  /* run the bulk, using streaming stores */
  for ( ; l_n < l_trip_stream;  l_n+=8 ) {
    _mm256_stream_pd( &(io_c[l_n]),   _mm256_loadu_pd(&(i_a[l_n]))   );
    _mm256_stream_pd( &(io_c[l_n+4]), _mm256_loadu_pd(&(i_a[l_n+4])) );
  }
  /* run the epilogue */
  for ( ; l_n < i_length;  l_n++ ) {
    io_c[l_n] = i_a[l_n];
  }
}
```

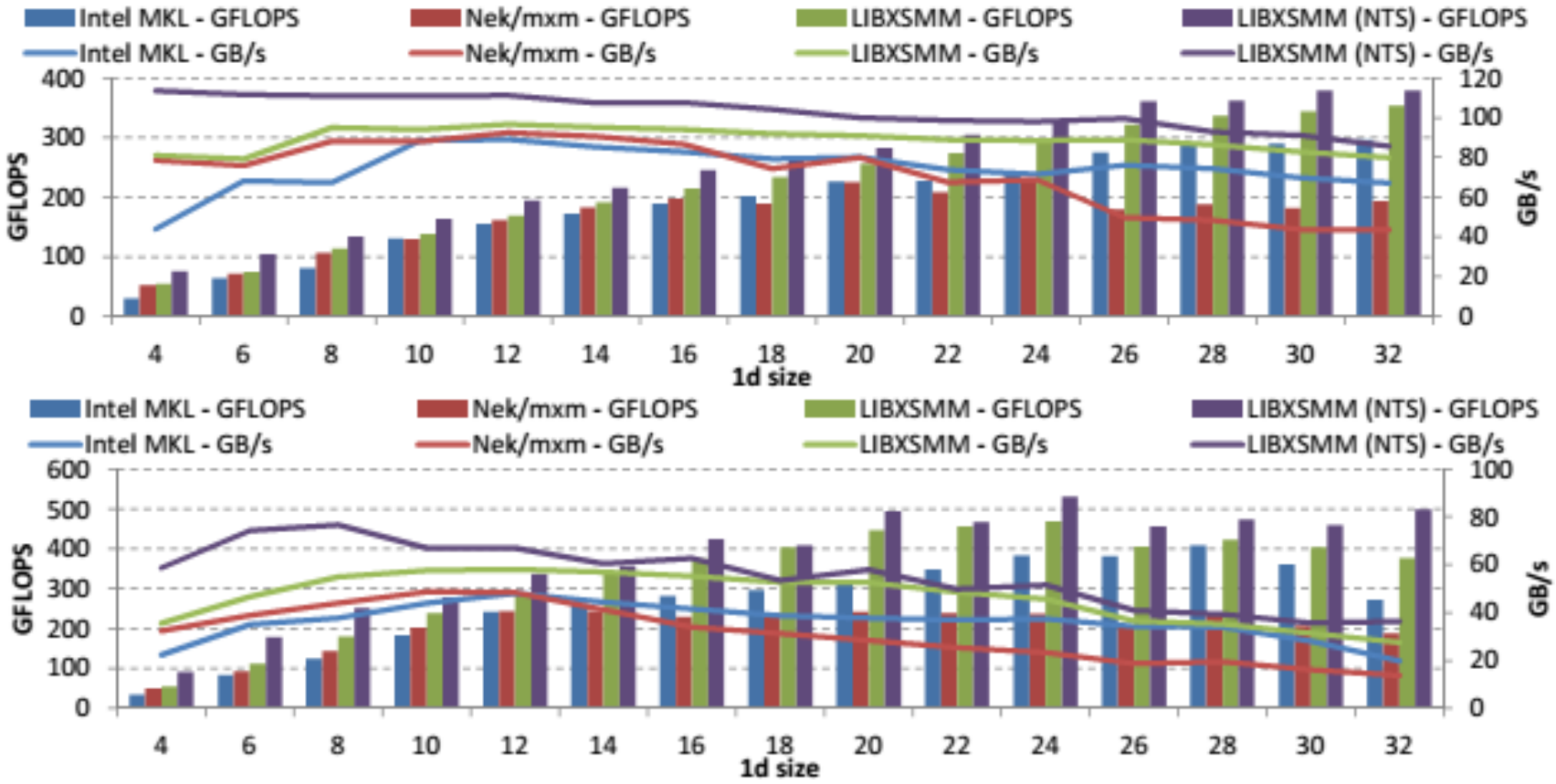
Systems we used in this Study

- Mira is a IBM BlueGene/Q with 49,152 nodes hosted at ANL in the US. Each node has 16 cores with 4 hardware threads per core and can support 204.8 GFLOPS and 30 GiB/s main memory bandwidth. (FLOP/BYTES ~ 6)

- Shaheen is a Cray XC40 with 6144 nodes hosted at KAUST in Saudi Arabia. Each node has two Intel® Xeon® E5-2698v3 (code-named Haswell) processors with 16 cores each and can support around 1177.6 GFLOPS and 101.6 GiB/s main memory bandwidth, combined on both NUMA domains. (FLOPS/BYTE ~10)

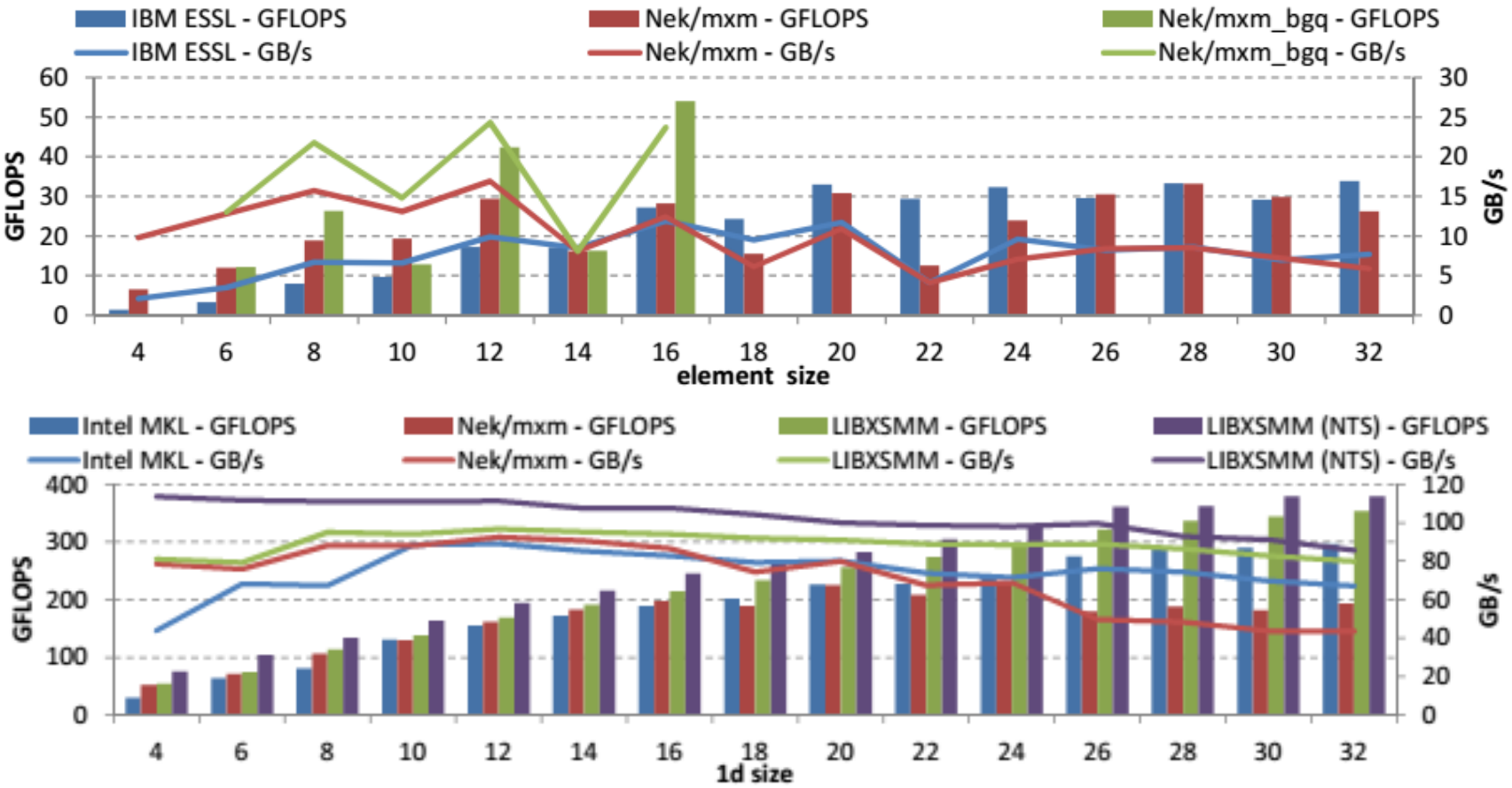
→ Shaheen's cores therefore have 2.9× the floating point and 1.7× the memory bandwidth of Mira's BlueGene/Q cores.

Helmholtz Op. and Basis Transf. on Shaheen



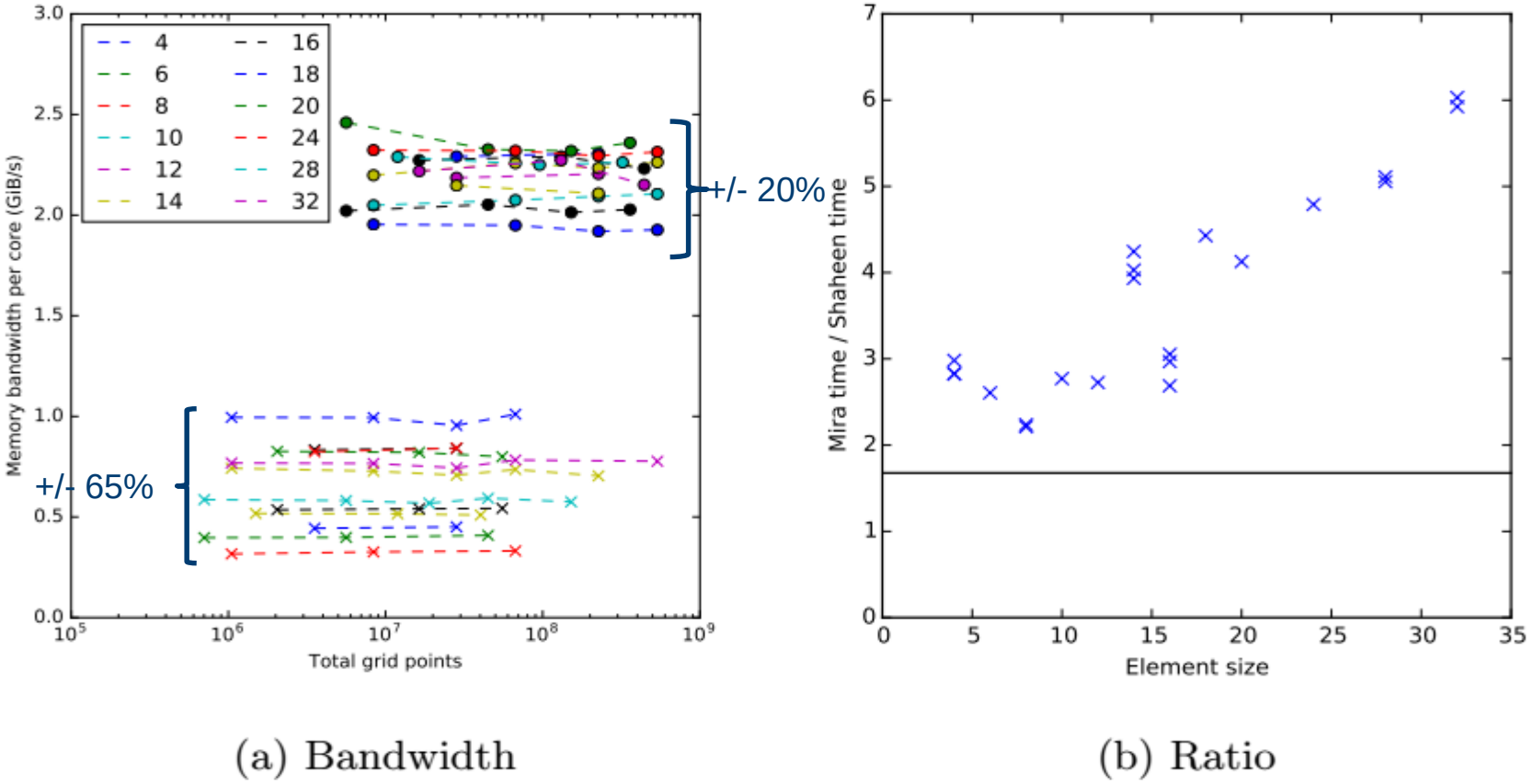
Performance of reproducers for the Helmholtz operator (top) and Basis Transformation (bottom) using different implementation for the small matrix multiplications. NTS denotes the usage of non-temporal stores. Measured on Shaheen (32 cores of HSW-EP, 2.3 GHz)

Helmholtz-Operator on Mira (BG/Q) vs. Shaheen



Performance of reproducer for the Helmholtz operator reproducer on Mira (top) and Shaheen (bottom). IBM Blue Gene/Q doesn't offer unaligned memory support for vector instructions and Intel Xeon benefits from more flexible hardware and its optimal utilization through LIBXSMM. mxm_bgq is an assembly library for Mira.

Identifying the optimal spectral order: Performance

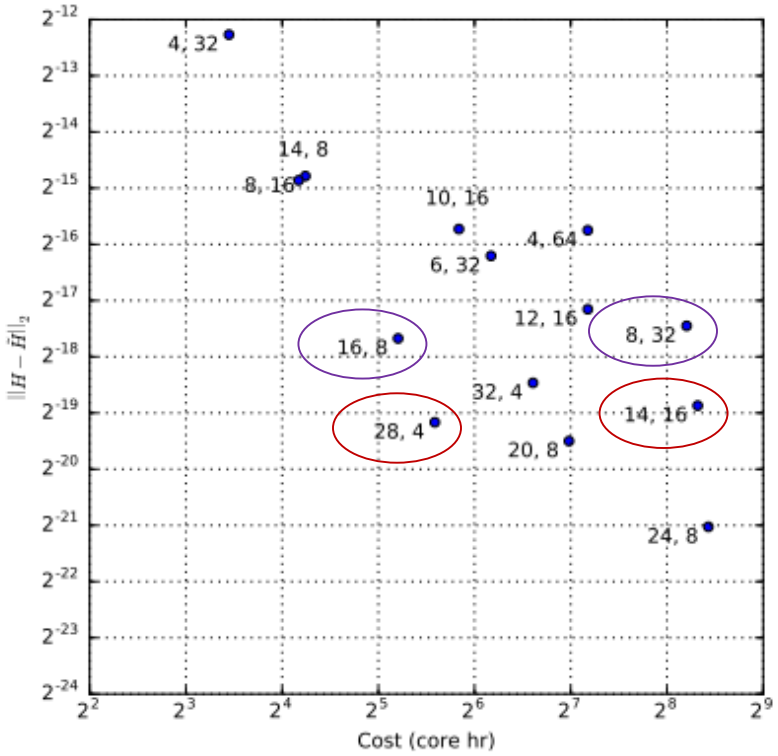


(a) Bandwidth

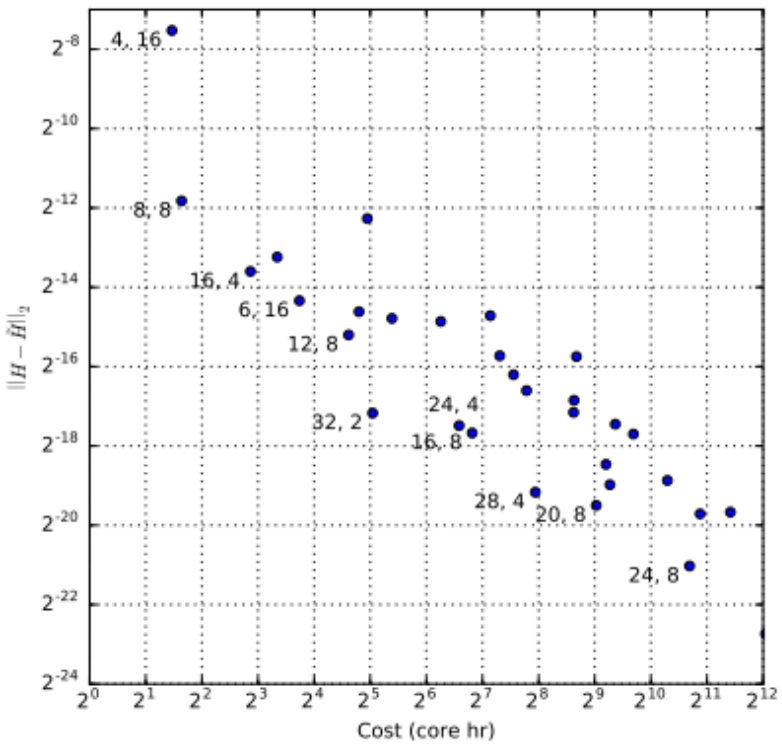
(b) Ratio

- Plot (a) confirms that all executions on Shaheen (circles) are bandwidth bound and Mira (crosses) is getting compute bound for higher orders
- This is due to the Xeon's stronger core + LIBXSMM which results into a significant speed-up wrt. time-to-solution as shown in Figure (b)

Identifying the optimal spectral order: Accuracy



(a) Shaheen



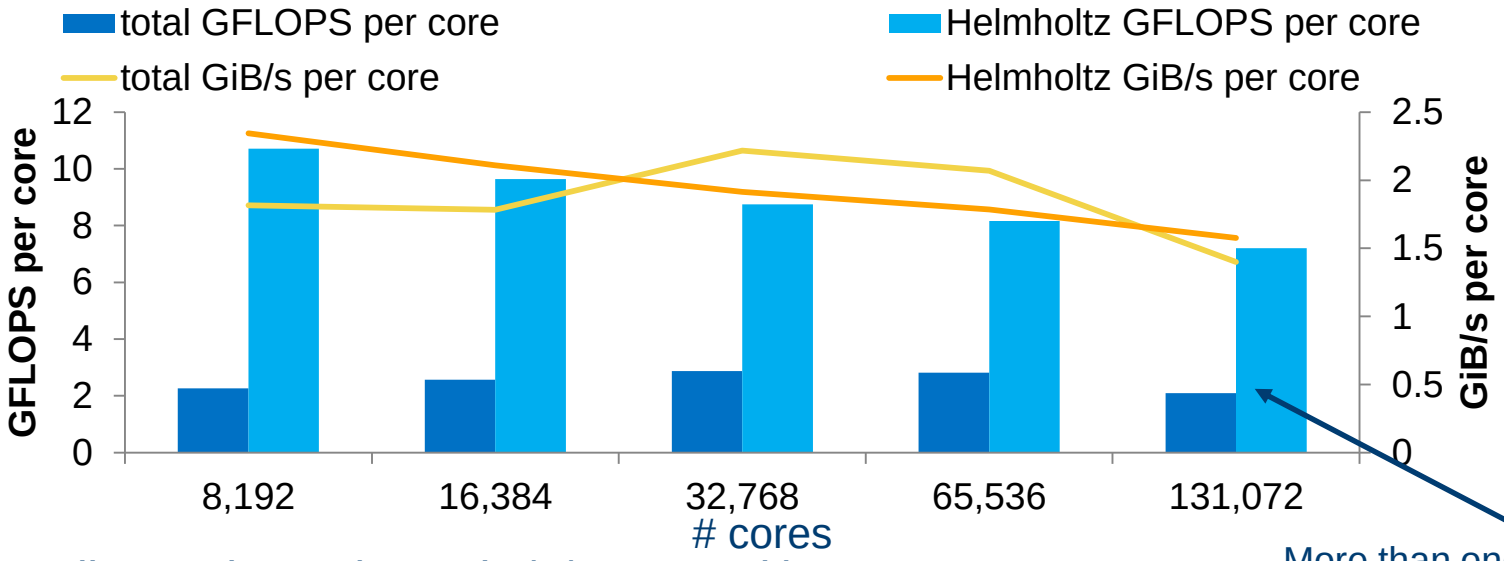
(b) Mira

Log-base 2 error of bubble height and mix volume on Shaheen and Mira, wrt. to time to solution.

We can see that high orders are favorable as they better match modern hardware and have superior convergence speed as they stay memory bandwidth bound for high orders.

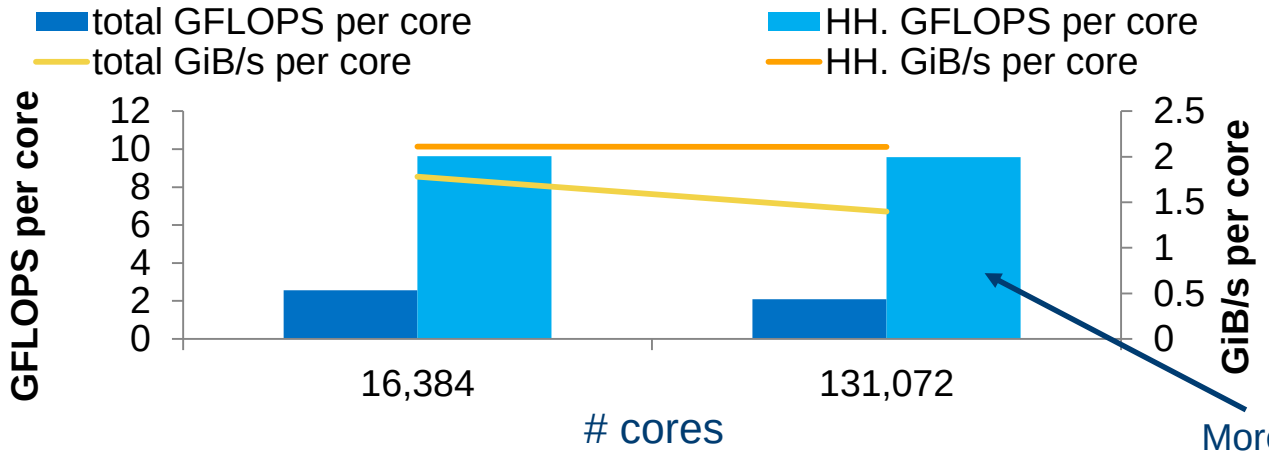
Performance at Petascale (Shaheen at KAUST)

Strong Scaling NekBox (3.1 GiB/s/core peak measured by stream TRIAD)



More than one PFLOPS

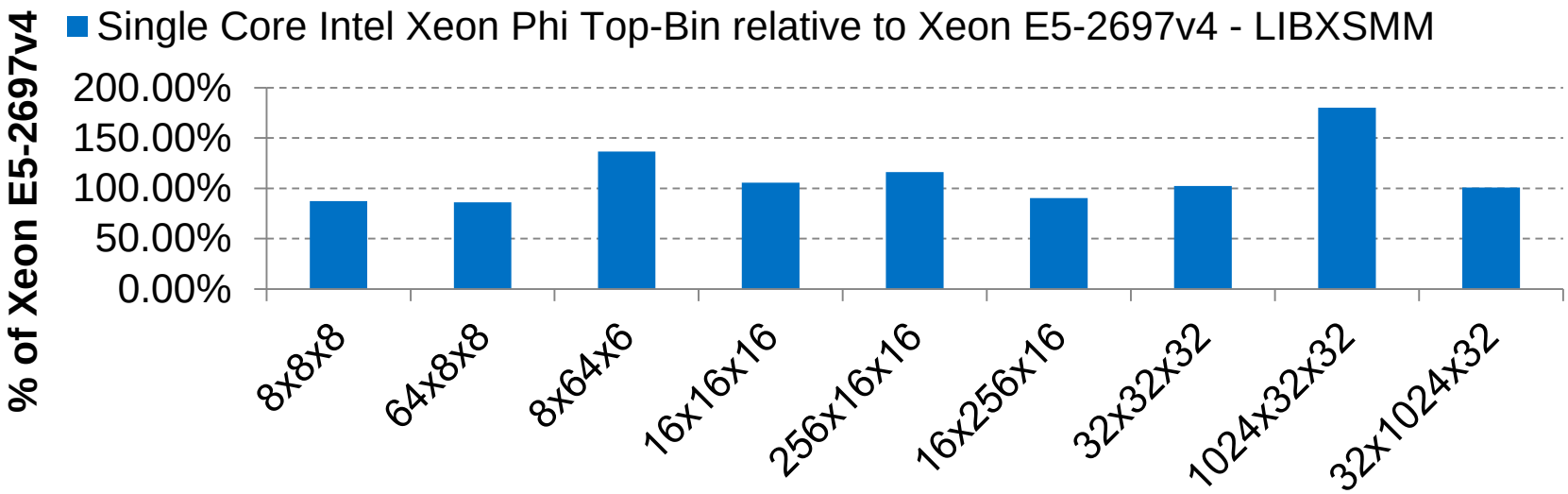
Weak Scaling NekBox (3.1 GiB/s/core peak)

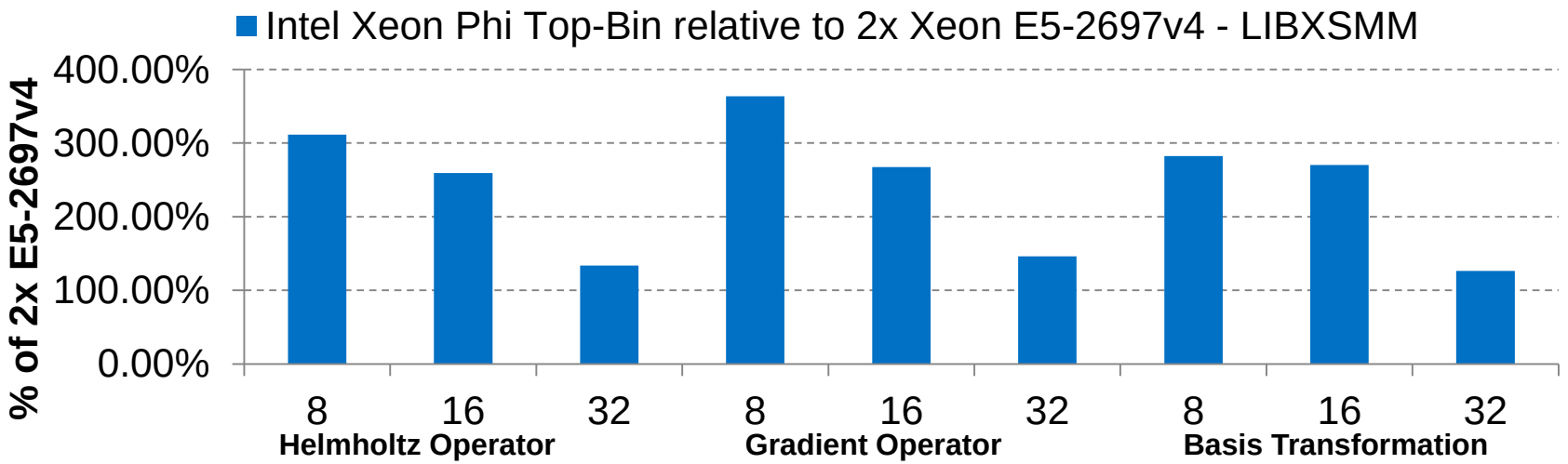


Excellent scaling for weak and strong

More than 1.3 PFLOPS

Xeon Phi 7250 (Knights Landing) results





Conclusion

- We showcase two keys for achieving on-node performance with Petascale Intel architectures:
  - Runtime code generation to accelerate the small GEMMs needed in SEM
  - Non-temporal stores to free memory bandwidth from RFOs
- Even with modern and powerful nodes, the Spectral Element Method is a good fit for large scale computing as it offers excellent weak and strong scaling properties
- These technologies allow us to push the efficiency frontier of SEM to very high orders where exponential convergence reduces computational cost
- Compared to Mira, Shaheen is up to 6x faster, outperforming the 2.9x difference in compute and 1.7x in bandwidth
  - Due to more efficient instructions and deep out-of-order execution with LIBXSMM on Intel architectures


intel®
experience
what's inside™