Optimization of Elastodynamic Finite Integration Technique on Intel Xeon Phi Knights Landing Processors

William C. Schneck, III
william.c.schneck@nasa.gov
NASA Langley Research Center
Hampton, VA 23681

Elizabeth D. Gregory
elizabeth.d.gregory@nasa.gov
NASA Langley Research Center
Hampton, VA 23681

Cara A. C. Leckey
cara.ac.leckey@nasa.gov
NASA Langley Research Center
Hampton, VA 23681

Abstract:

This work describes the development and optimization of an implementation of an isotropic elastodynamic finite integration technique (EFIT) code for parallelized computation on Intel Knights Landing (KNL) hardware. EFIT is a numerical approach resulting in standard staggered-grid finite difference equations for the elastodynamic equations of motion to simulate bulk waves is solids. The computationally efficient simulation of elastodynamic wave propagation and interactions in aerospace materials is of high-interest in the fields of nondestructive evaluation (NDE) and structural health monitoring (SHM). Ultrasonic inspection uses an ultrasonic signal, generated at the surface of the material/structure via use of a piezoelectric transducer, to propagate sound waves into the material where it interacts with any existing defects, as well as with structural boundaries and any material inhomogeneity. Reflections from defects and boundaries are then measured by a transducer. Realistic ultrasound simulation tools can significantly aid the development and optimization of inspection techniques and can assist in the interpretation of experimental data.

The optimization of an elastodynamics simulation code for the KNL Many Integrated Core processor was performed. The optimization focused on data locality and vectorization. Results show that tiling of the data to exploit the cache behavior and allow for significant utilization of the KNL hardware. The MPI implementation allows for a scalable implementation enabling large problems to be simulated. The model results were validated against theoretical dispersion curves to within 2% of the group velocity, and within 0.5% of the phase velocity of the A0 mode. Aggressive use of tiling, threading, and vectorization techniques allowed for dramatically improved time to solution.

1.0 Introduction

Trends in high performance computing have led to large increases in the degree of parallelism available to solve large problems more quickly. With these trends in hardware, researchers have been developing sophisticated implementations of a broad suite of existing algorithms (such as discontinuous Galerkin methods [1], mesh partitioning schemes [2], or other partial differential equation techniques [3]) for a wide variety of physics including computational fluid dynamics [4], elastodynamics and acoustics [3], plasma simulation [5], and astronomy [6], to make better use of the new hardware. Many Integrated Core (MIC) computing architectures are creating opportunities to achieve parallelism within a compute node by increasing both the on-chip and in-core level of parallelism, while creating lower cost computing options compared to traditional CPU cluster computing. The Intel Xeon Phi codenamed Knights Landing (KNL) processor family became available to the public starting in mid-2016 [7] and is a many-core hardware architecture with 64, 68, or 72 CPU cores on a single package. KNL is the second-generation Intel Xeon Phi MIC device [8]. The first generation device (Knight's Corner, KNC [9]) operates as a coprocessor, so calculations are off-loaded to the KNC from the host computer. Alternatively, KNL can work as a bootable CPU, enabling far easier interrogation of code operations for de-bugging and optimization. KNL can also run Intel Xeon binaries, thus reducing the initial development difficulties compared to KNC. This paper explores methods in both memory management and code vectorization to exploit the use of KNL cores for optimized elastodynamic simulations.

The computationally efficient simulation of elastodynamic wave propagation and interactions in aerospace materials is of high-interest in the field of nondestructive evaluation (NDE) and structural health monitoring (SHM). Numerous industries (e.g., automotive, aerospace, infrastructure, oil and gas), rely on the use of NDE/SHM methods to ensure reliability and safety of structural components (e.g., airplanes, spacecraft, bridges, railways, oil pipelines). Current inspection methods utilize various physics based techniques, including use of elastodynamic waves (e.g., ultrasound inspection), heat flow (e.g., thermography inspection), and electromagnetic waves (e.g., terahertz imaging), among others. The use of elastodynamic waves in solid materials at ultrasonic frequencies is one of the most common inspection approaches currently used in many industries. While ultrasound is defined as spanning the frequency range of 20 kHz to 1 GHz, use of frequencies between 100 kHz and 20 MHz is common practice.

In the most common ultrasonic inspection setups an ultrasonic signal is generated at the surface of the material/structure via use of a piezoelectric transducer. The ultrasonic wave propagates into the material in the form of ultrasonic bulk waves (shear and longitudinal). The waves interact with any existing defects, as well as with structural boundaries and any material inhomogeneity. Defects and boundaries can lead to reflection of the ultrasonic wave back to toward the surface, where it is measured by the transducer. This general concept is outlined in the diagram in Figure 1(a). Bulk wave ultrasound methods are widely used in the aerospace industry for inspection of aircraft and spacecraft components.

In addition, an approach frequently investigated for ultrasonic inspection of pipe or plate-like structures (e.g., oil pipelines, aircraft components) is the use of ultrasonic guided waves, also known as Lamb waves. Lamb waves are a physical phenomenon arising from the presence of a wave-guide (i.e., upper and lower plate boundaries). The wave-guide leads to coupling of shear and longitudinal ultrasonic waves to form discrete wave modes which propagate along the length of the plate with different displacement profiles across the plate thickness. Symmetric (extensional) and antisymmetric (flexural) guided wave modes are two fundamental mode types that can exist in an isotropic plate. Each guided wave mode has group and phase velocities determined based on the frequency of the transducer excitation and the plate thickness. In guided wave methods it is common to use one transducer to excite ultrasonic guided waves in the plate and receive resulting signals with a second transducer. The presence of damage can lead to reflected ultrasound waves, forward scattered waves, changes in mode velocity, and mode conversion [10]. A diagram showing the general concept of ultrasonic guided waves is shown in Figure 1(b). More details on ultrasonic bulk and guided wave methods can be found in [11].

Guided wave damage detection and quantification techniques reported in the literature range from sparse data methods via sensor arrays to dense data methods utilizing full wavefield imaging through laser Doppler vibrometry. Numerous authors have reported on experimental guided wave techniques and simulation-aided investigation of guided wave methods [12] [13] [14] [15] [16] [17] [18].

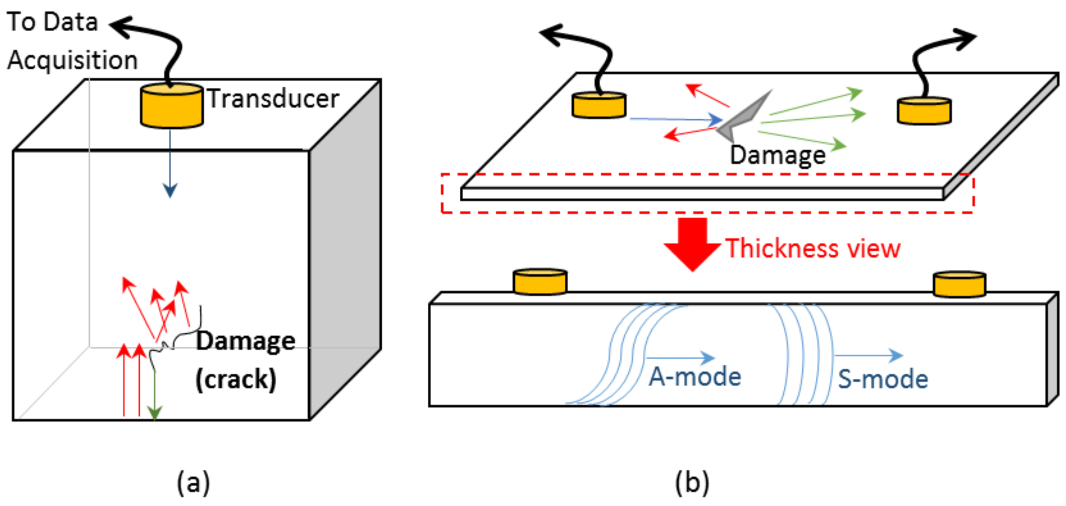


Figure 1: Diagram showing two different typical ultrasonic inspection scenarios: (a) pulse-echo ultrasound setup creating ultrasonic bulk waves, (b) ultrasonic guided waves in a thin plate component. In both (a) and (b), blue arrows show the ultrasonic wave propagation direction generated by the transducer, red arrows show backward reflected ultrasound waves, and green arrows represent forward scattered ultrasound waves.

For both bulk and guided wave based ultrasound methods, realistic ultrasound simulation tools can significantly aid the development and optimization of inspection techniques and can assist in the interpretation of experimental data. Some common numerical simulation methods for ultrasonic wave modeling include finite element analysis (FEA), finite difference and boundary element methods, among others. The elastodynamic finite integration technique (EFIT) is a numerical approach resulting in standard staggered-grid finite difference equations for the elastodynamic equations of motion. EFIT has been in use since the 1990s [19] [20]. More recently, the method has been used for 3D simulation of ultrasound propagation in metallic and composite aerospace components [10] [16] [21] [22] [23]. The method is a second order accurate, explicit time domain leap-frog finite difference scheme performed on a structured cubic grid, with a 6-point stencil. The simplicity of the mathematical scheme makes the method readily portable to various parallelization schemes and computational hardware.

Optimization of elastodynamics schemes has been undertaken for various architectures ranging from conventional CPU clusters to MIC and GPU (graphical processing unit) architectures [24] [25] [26]. Huthwaite performed optimizations for unstructured grid elastodynamic explicit FEA on GPUs [25]. Particularly, that work addressed data arrangement in memory to optimize accesses for locality to best use the GPU compute capability. Huthwaite demonstrated improved data blocking in memory, so that loads to the shared cache were unlikely to be evicted before calculations were complete. Castro et.al. focused on seismic wave propagation simulations on Intel KNC, NVIDIA K20, Kalray MPPA-256, and Intel Xeon E5-4640 (Sandy Bridge) [26]. The objective of that broad based comparison was to perform some optimizations of the EFIT algorithm for each investigated platform, and evaluate the compute efficiency in terms of both time and energy usage. The optimizations included data tiling and vectorization to improve cache re-use and compute speed. Castro, et.al. observed a compute time per cell per time step of $8.85\ ns$ for a simulation of 180^3 (6 million) cells for 500 time steps.

The purpose of this work is to develop an optimized implementation of isotropic EFIT for parallelized computation on Intel KNL hardware. The targeted applications are for thin structures more resembling aerospace structures. This results in larger numbers of cells and time steps to adequately resolve the ultrasound passing through the simulated material. Aggressive use of tiling, threading, and vectorization techniques allowed for dramatically improved time to solution.

The above mentioned increase in on-chip and in-core parallelization enables a system capable of increased floating point operations per second (FLOPS) without requiring sophisticated cooling. This reduces the power cost per floating point operation, thus enabling a greater number of total floating point operations within the chipset. Additionally, this 'network-on-a-chip' architecture allows for large-scale task parallelism that is interconnected by the on-chip bus, rather than over the system network. The on-chip memory has significantly higher bandwidth than the system memory (MCDRAM bandwidth ≈400 GB/s vs DRAM bandwidth ≈90 GB/s), thus helping to improve the efficiency of algorithms that are traditionally bandwidth bound. A diagram of the KNL device can be seen in Figure 2. Each tile is a heavily modified dual-core 14nm Airmont processor connected into a high-bandwidth 2D mesh interconnect. The Airmonts have been modified to include two 512-bit vector processor units (VPU), allowing up to 8 double precision operations per VPU per clock cycle. The dual-core tiles share a 1MB L2 cache, and each core has its own 32kB L1 data cache (a 32kB L1 instruction cache is present as well). A far more comprehensive discussion of the design and microarchitecture of the KNL can be found in [8].

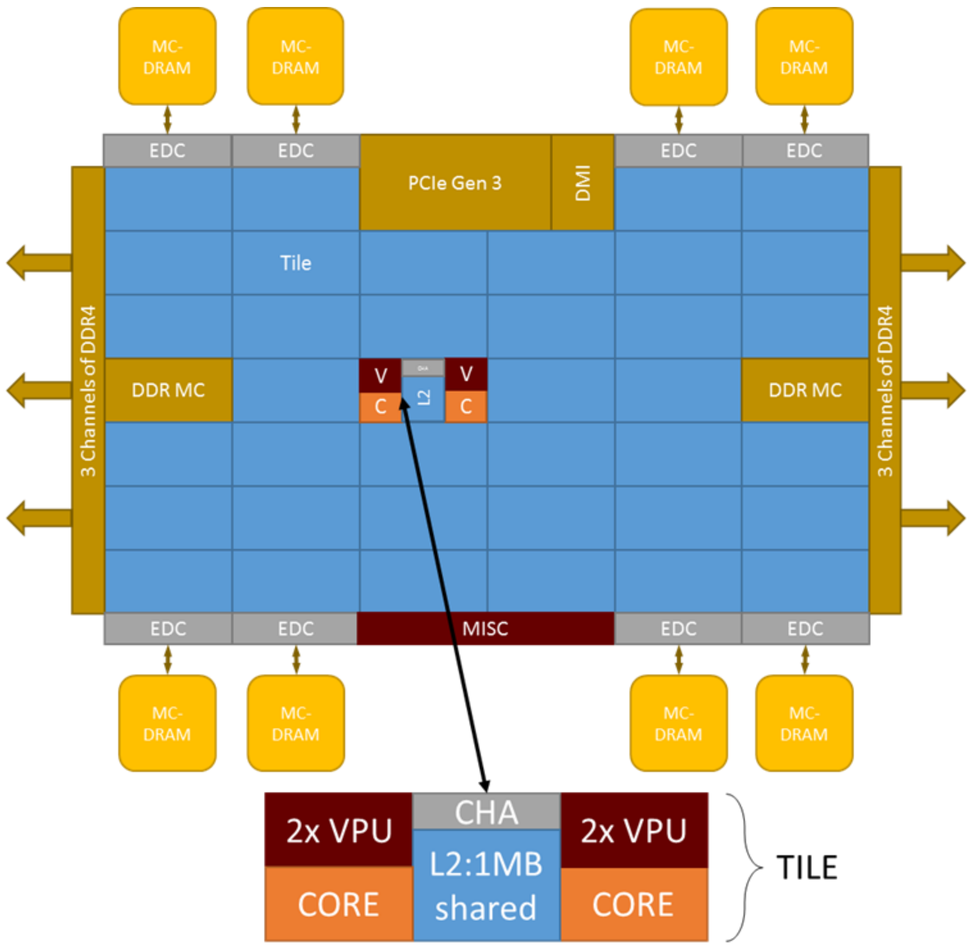


Figure 2: KNL hardware diagram showing the processor tiling and connections to on-package (Multi-Channel (MC) DRAM, EDC is the MCDRAM controller) and off-package (DRAM DDR4) memory (diagram based on [8]).

Section 2.0 describes the EFIT mathematical approach as a basis to discuss the computational implementation. Section 3.0 describes the developed KNL-specific numerical implementation approach and studies on memory layout optimization. Section 4.0 shows results for scalability and timing tests, as well as results of roofline studies. Section 5.0 discusses a code validation case for the KNL EFIT implementation and Section 6.0 describes an example use case relevant to ultrasonic NDE. Last, Section 7.0 summarizes the findings in this study and discusses areas for future work.

2.0 Mathematical Formulation

The elastodynamic equations of motion are presented in Equations (1) and (2):

$$\rho \frac{\partial v_i}{\partial t} = \sum_j \left(\frac{\partial \sigma_{ij}}{\partial q_j} \right) + f_i \qquad \textbf{(1)}$$

$$\frac{\partial \sigma_{ij}}{\partial t} = \frac{1}{2}\sum_k \sum_l \left(c_{ijkl} \left(\frac{\partial v_k}{\partial q_l} + \frac{\partial v_l}{\partial q_k} \right) \right) \tag{2}$$

where σ_{ij} is the stress tensor, v_i is the velocity vector, ρ is the material density, f_i is the applied force, q is a generalized Cartesian coordinate, and c_{ijkl} is the material compliance tensor. The EFIT mathematical kernel is written in C with use of MPI+OpenMP. These equations are discretized on a 'forward' staggered grid with the normal stresses located at the cell centers, the velocity components located at their relevant normal faces (e.g., v_y located at the 'upper' xz plane), and the shear stresses located at their relevant 'forward' edge (e.g., T_{xy} located that the center of the edge defined by the 'upper' xz plane and the 'forward' yz plane). Figure 3 shows this forward stagger.

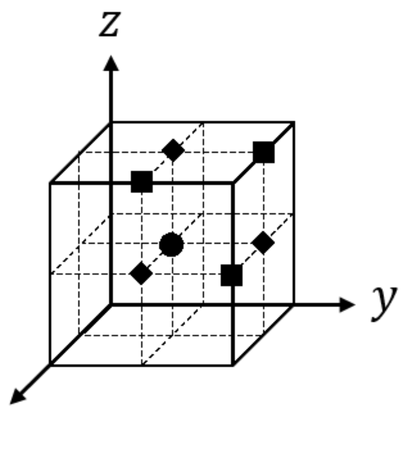


Figure 3: Depiction of the staggered grid used in the EFIT algorithm. Squares indicate locations where shear stress is evaluated, diamonds indicate locations where velocities are evaluated, and the circle is located where all three normal stresses are evaluated.

The discretized EFIT equations for these equations of motion are presented in Equations (3), (4), and (5):

$$v_i^{n,m+1} = v_i^{n,m} + \frac{2\Delta t}{\rho^n + \rho^{n+q_i}} \left[\frac{\left(S_{ii}^{n+q_i,m+\frac{1}{2}} - S_{ii}^{n,m+\frac{1}{2}} \right) + \sum_{j \neq i} \left(T_{ij}^{n,m+\frac{1}{2}} - T_{ij}^{n-q_j,m+\frac{1}{2}} \right)}{\Delta q} + f_i^n \right] \tag{3}$$

$$S_{ii}^{n,m+\frac{1}{2}} = S_{ii}^{n,m-\frac{1}{2}} + \frac{\Delta t}{\Delta q} \left[(\lambda^n + 2\mu^n)\left(v_i^{n,m} - v_i^{n-q_i,m} \right) + \lambda^n \sum_{j \neq i} \left(v_j^{n,m} - v_j^{n-q_j,m} \right) \right] \tag{4}$$

$$T_{ij}^{n,m+\frac{1}{2}} = T_{ij}^{n,m-\frac{1}{2}} + \frac{4\Delta t \left[\left(v_i^{n+q_j,m} - v_i^{n,m} \right) + \left(v_j^{n+q_i,m} - v_j^{n,m} \right) \right]}{\Delta q \left(\frac{1}{\mu^n} + \frac{1}{\mu^{n+q_i}} + \frac{1}{\mu^{n+q_j}} + \frac{1}{\mu^{n+q_i+q_j}} \right)} \tag{5}$$

where n is the array index, operations between n and q indicate movement in the q direction in the array, m indicates time index, S_{ii} are the normal stresses, T_{ij} are the shear stresses, Δt is the time step size, Δq is the spatial step size, and λ and μ are the first and second Lamé parameters. These equations are based on the method described in [19] [20].

3.0 Numerical Implementation

Efficient implementation of simulation code is essential to achieving the required time to solution for effective forward modeling for NDE/SHM applications. Forward model simulations of the physics of ultrasound inspection can enable prediction of part inspectability and cost-effective inspection method optimization. Decreasing simulation time while maintaining an accurate representation of the inspection problem is also particularly important for parameter estimation (e.g., reflector size, shape, and location estimation), due to the requirement to make many executions of the forward model in solving the inverse problem. This section describes the methodology used for EFIT optimization on KNL hardware.

3.1 KNL configuration

The optimizations were developed for an Intel Xeon Phi 7210 KNL configured in quadrant/cache mode. Quadrant mode maintains data locality, reducing cache miss latency and bandwidth penalties, without requiring NUMA (non-uniform memory access)-aware programming. Cache mode sets the 16GB of on-package high bandwidth memory (MCDRAM) to function as a last level cache (LLC), thus allowing use of the MCDRAM without requiring further NUMA-aware programming. The large amount of data that is required for the simulation makes using the MCDRAM as a cache attractive, allowing rapid access to a large amount of data prefetched from system memory into this LLC. The various configurations of the KNL are discussed more in-depth in [8].

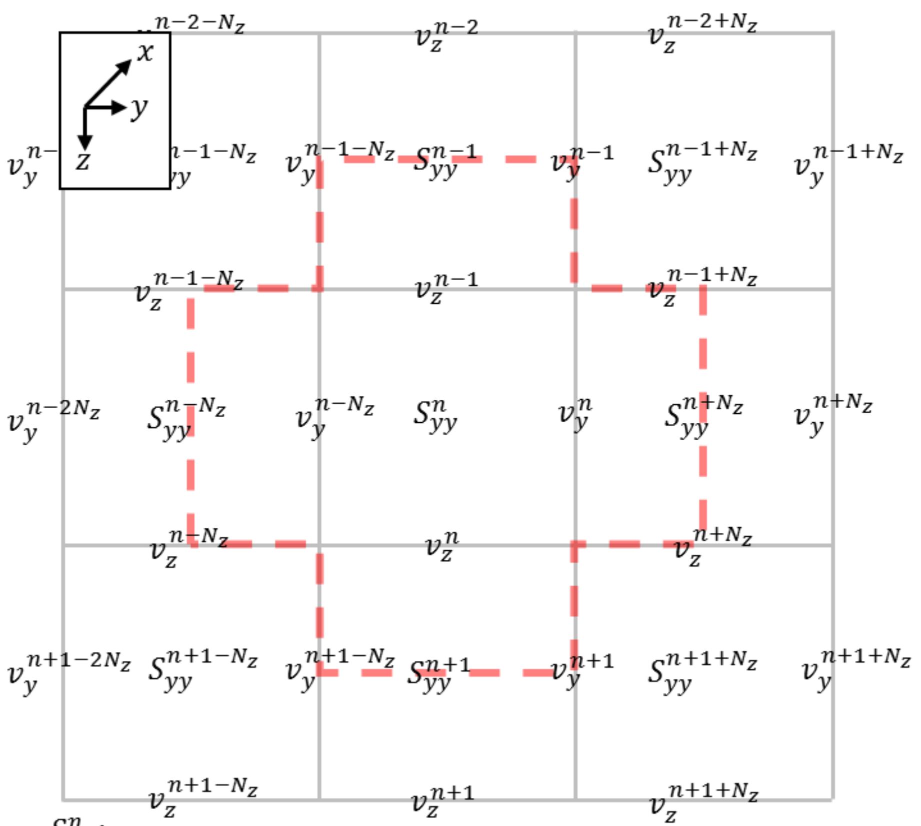
3.2 Implementation

The computational implementation of mathematical equations must effectively use the execution hardware in order to produce useful data in a timely fashion. For the implementation of the EFIT algorithm on KNL hardware in particular, this means leveraging the VPUs to compute stencil calculations while appropriately managing internal stress-free boundaries associated with simulating internal damage (such as a crack). The following subsections discuss a vectorization approach for 'material' grid cells, as well as an approach for dealing with grid cells associated with stress-free boundaries.

3.2.1 Vectorization and Stencil

The EFIT algorithm is evaluated on a cubic grid. The individual operations associated with evaluating a given state are highly vectorizable, as each non-boundary grid point is mathematically/behaviorally identical. As long as a structures-of-arrays programming approach is used, the VPUs are efficiently utilized by a unit-stride access. The KNL compute cores can perform up to 16 double precision calculations per operation if full utilization of the VPU's is achieved. VPU's allow for the implementation of a set of instructions operating on one-dimensional arrays (rather than operating on single elements). Failure to use the vector registers can adversely impact simulation performance by a factor of up to 16. Thus, any code intended for use on a KNL device should be programmed with a 'vector aware' approach. Further, to reduce the time spent moving data between levels of the memory hierarchy, each vector going into the calculation should have a unit stride. Failure/inability to use a unit stride can result in gather/scatter operations which, depending on the size of the stride, will greatly reduce performance. Failing to use a unit stride access through the data also leads to increased communications traffic between levels of the memory hierarchy.

For regular grids such as for the EFIT finite difference calculations, achieving unit stride is fairly straight forward since the calculations to solve for variables at the neighboring spatial grid point (in memory sequence) only need to access data that is offset by a unit stride. This concept is demonstrated for a set of 2D finite differences in Figure 4. Each velocity calculation shown in Equation (3) is performed for each grid cell using the stress values from the prior time step. The data required for each velocity calculation is ordered sequentially such that to advance from one spatial cell to the next always increments the array access by one. Next, the same pattern is followed for the stress calculations, using the velocity calculations from the current time step. The time step is then incremented forward. The array access pattern is demonstrated in Figure 4 where, as before, v_y and v_z are the y and z direction velocities and S_{yy} is the y direction normal stress. The figure shows the example of taking differences across a cell for both the y and z directions. All evaluations at $n+1$ are the same as the evaluations at n, with all indices incremented by one. As each index in the $n+1$ equations have been incremented by one, this demonstrates a "well vectorized" operation. Figure 4 shows the relevant difference calculations across the cell, and for the next cell in the calculation sequence. Depending on the access stride between y and $y+1$, e.g., if N_z is not an integer multiple of the cache line size, there may be memory alignment penalties for the y direction differences, and there are almost certainly memory alignment penalties in the z direction differences. Memory management considerations are discussed further in the next section.



S_{yy}^{n}:

$$\left.\frac{\partial v_y}{\partial y}\right|_n = \frac{v_y^n - v_y^{n-N_z}}{\Delta q} \quad , \quad \left.\frac{\partial v_z}{\partial z}\right|_n = \frac{v_z^n - v_z^{n-1}}{\Delta q}$$

S_{yy}^{n+1}:

$$\left.\frac{\partial v_y}{\partial y}\right|_{n+1} = \frac{v_y^{n+1} - v_y^{n+1-N_z}}{\Delta q} \quad , \quad \left.\frac{\partial v_z}{\partial z}\right|_{n+1} = \frac{v_z^{n+1} - v_z^{n+1-1}}{\Delta q}$$

Figure 4: 2D stencil of difference calculations showing the array access pattern for velocity differences at current spatial grid cell n, and the next grid cell (in memory sequence), $n+1$, where the column stride is represented by N_z. These operations are readily vectorized.

3.2.2 Boundary Conditions

For NDE applications, it is often sensible to implement stress-free boundaries at the edges of the simulation space. Additionally, defects/damage inside of the simulated specimen are frequently represented by the specified locations of stress-free boundaries interior to the simulation (for example, crack damage or disbonds). Boundary conditions must be evaluated in a way that they cause minimal disruption to the vectorized interior region. The evaluation of boundary conditions is complicated by the inclusion of boundaries associated with damage internal to the material region. This challenge is addressed by a technique called masking. In a masking scenario the vector registers perform the same calculations on all grid points, but a 'mask' prevents the results from points with the incorrect boundary/non-boundary equation from being stored/integrated into the solution. An example of a masked vector operation is shown in Figure 5.

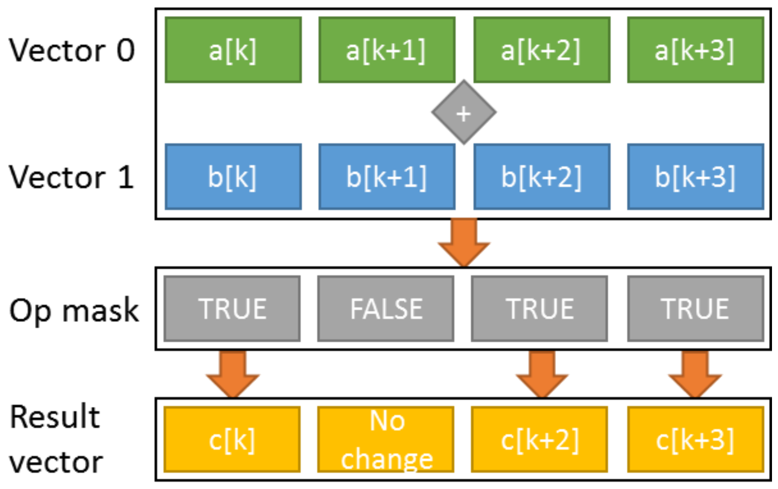


Figure 5: Example masked vector operation. The diagram shows that where the mask value is 'false', the vector result is not stored to the output vector register.

In general, boundary conditions require special consideration in vectorized codes. Boundary cells must be handled differently from interior calculations. Achieving clustering of exterior boundary cell data in hardware memory allows some of the boundary cells to operate together in a vectorized manner, however this approach does not necessarily allow for reasonable implementation of interior boundaries. This challenge is a critical problem for NDE as the simulations are carried out to understand wave interaction with damage contained within the material. Thread branching/divergence rapidly undoes vectorization, therefore it is critical to evaluate conditionals outside loops, where possible (e.g. if the x index is zero, evaluate the $-x$ boundary condition). The KNL architecture does allow for masked vector operations for conditionals that by necessity must be evaluated at each grid element. If conditionals are able to be phrased as a mask, the operation will remain vectorized, and will not store unneeded operations. Excessively masked operations, however, begin to approach non-vectorized performance ceilings. If the damage within a simulated specimen are localized rather than being distributed damage (which is often the case for localized crack damage in metals), the masking will allow the operations to continue with minimal performance penalties for the majority of the simulation domain.

3.3 General Data Layout

Data layout in computing hardware memory significantly affects the performance of modern scientific software. 'Network-on-a-chip' hardware architectures like the KNL have significant bandwidth advantages compared to cross-socket and inter-node communications. However, the KNL bandwidth to on-chip MCDRAM is still much slower than L1 or L2 cache access. For communication bound problems, maximizing access hits from L1 or L2 cache requests is required to achieve effective code speedup. To maximize L1 and L2 cache use, more sophisticated data layout in hardware memory is needed to enable greater cache re-use, and therefore reduce pressure on the (already heavily loaded) communications interconnects.

For simulating ultrasound in plate-like components, the geometry being modeled is typically small in one dimension (e.g., plate thickness, z direction) and large in the other two dimensions. Communication across threads and processes generally incurs greater overhead, so the memory layout should be designed to reduce this communication (i.e. maintain data locality). Schematically, to avoid additional communication across threads in the z dimension, any memory layout scheme should preserve the continuity of a page of data in the short dimension. Thus, thread and process boundaries should only be stenciled in the long dimensions. A memory layout optimization study (with the z direction small and x and y directions large) is described below.

A technique called 'tiling' is commonly used to subsection computational domains into L1 or L2 'friendly' chunks [27]. These chunks can frequently be transmitted in fewer, larger messages, allowing for reduced latency penalties. Furthermore, because these chunks are almost entirely self-contained within a tile, there are far fewer repeated transfers of data within the memory hierarchy. Thus, data tiling reduces load on the interconnect. Approaching this problem with 'cache aware' tiling permits the design of data layouts which can be optimized. A data layout optimization study for EFIT is discussed in Section 4.1.

The optimization study is conducted with the limitation that the layout requires unit-stride in the z direction. Strides in the x and y directions are tiled to seek an optimal data locality arrangement. This approach allows interrogation/exploration of the parameter space by varying the 'width' of the data pages, which are then stacked in the x direction. These stacks are then located side by side, creating a second index in the y direction, that will be referred to as the 'column index'. Figure 6 shows how an MPI process is broken up for two memory layouts: a naïve layout, and a more optimal layout. The naïve approach entails directly mapping the data to memory with no tiling, resulting in excessive memory contention and poor cache re-use. The more optimal layout tiles the data, significantly improving data locality (which reduces memory contention), and cache reuse.

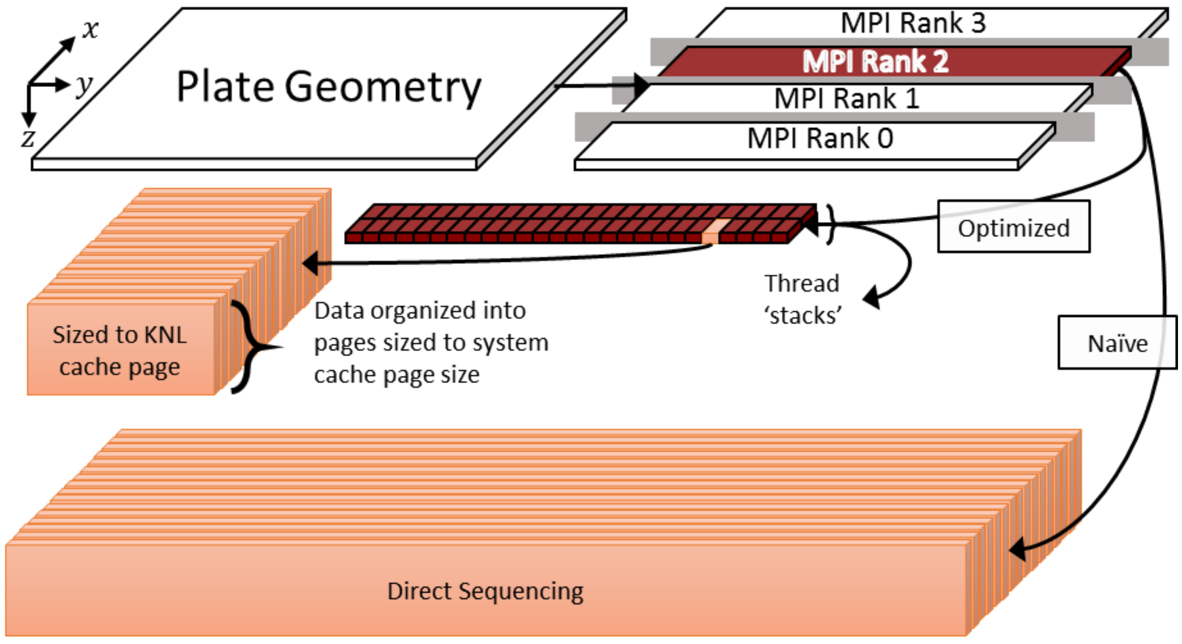


Figure 6: Diagram showing the how the data layout in memory is related to simulated plate. The simulated plate geometry is broken up along the x direction using MPI for necessary data passing. Within each MPI rank, the data are laid out in stacks of 2D tiles of data, enabling each thread to perform operations on its local data, reducing contention with neighboring threads for memory accesses. Note that the case where the page is set to a width of one or a width of the plate results in "Naïve" (directly mapped/sequenced) memory layouts.

4.0 Performance Studies

In this section a data layout optimization study will be presented, along with three code performance assessment methods. The data layout in memory will optimize the tiling of the data based on the time-to-solution for several simulation cases using different tile widths. The performance of the code running under the optimized data layout is assessed using roofline, weak scaling, and strong scaling analyses. Roofline analysis is a standard code analysis approach that is a means to determine how effectively a given code or kernel uses the available compute resources. In particular, it considers in a reasonably holistic sense, the hardware system and the algorithm in use, to give information regarding remaining gains that can be achieved in code speedup. The roofline analysis provides clues on ideal targets for further code optimization (for example, if the execution is found to be communication bound, one would not target improvement of execution hardware utilization). Weak and strong scaling describe the parallel performance of the algorithm. Weak scaling determines for a specified time to solution (i.e., execution time), how 'big' of a problem can be solved when more processors are added to perform the calculation. Weak scaling is important for problems where the desired computational domain is large, and small scale tests execute sufficiently quickly. In such a scenario, good weak scaling indicates that more hardware allows the solution of the larger computational domain in approximately the same execution time. Strong scaling demonstrates, for a given computational domain size, whether more processors working the calculation will yield faster time to solution. Strong scaling is important when the simulation domain is fixed, but improvements in actual time to solution are desired.

4.1 EFIT Data Layout Optimization

As discussed in Section 3.3, data layout in memory is critical to code performance. The objective of this data layout optimization study for EFIT is to optimize the layout of the simulation domain in memory in order to achieve the greatest possible cache reuse. This goal is achieved by creating stacks of 2D tiles (yz tiles stacked in the x direction). When the size is optimized, calculating all values in a tile before moving to the next tile provides greater cache reuse, and thus reduced pressure on the processor tile interconnect.

Data layout experiments were run in order to experimentally determine the optimal data layout in memory to permit best cache reuse. The results are shown in Figure 7. Simulation sizes of 4 million (512×512×16), 17M (1024×1024×16), 67M (2048×2048×16), and 270M (4096×4096×16) grid cells were studied. These sizes were selected to provide a broad suite of problem sizes to assess data layout sensitivities to changes in problem size. There was observed minimal sensitivity to problem size, except for problems that in large part do not fit in MCDRAM.

The data layout optimization study was conducted by changing the tiled stack width, as shown in Figure 6, from a page width of unity to the 'direct sequencing' arrangement, by powers of two. Figure 7 shows the results of memory layout experiments. As shown in the figure, for problem sizes of 4M to 67M grid cells, which fit within the high-bandwidth MCDRAM, optimal data arrangements occur for data pages sized close to the size of the cache page. For problems large enough to be worked from the 'far' DRAM memory, the optimization is less sensitive to the tile size. However, for large problem sizes, improvement is still observed at, or slightly larger than, data pages sized to the cache page size (i.e., problems worked from DRAM are still aided by the tiling optimization though not as significantly as for problems worked from MCDRAM or L2). These layouts were studied for single-device as well ask up to four InfiniBand connected KNL devices communicating through MPI.

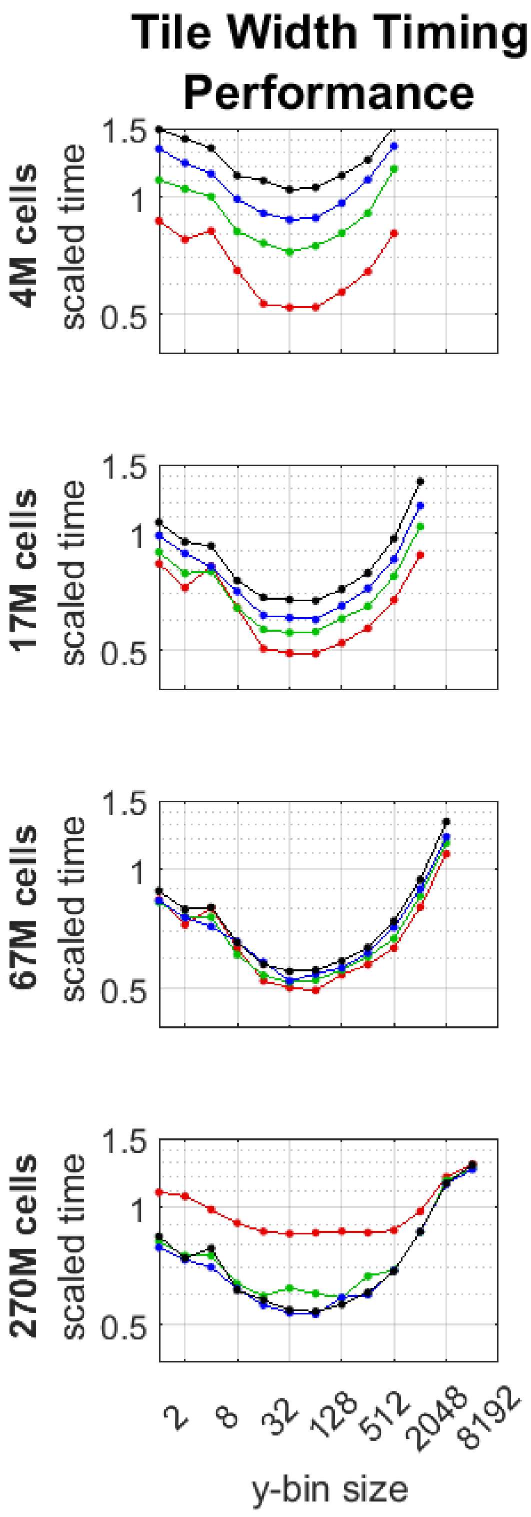


Figure 7: Data layout experiments. The red lines are for 1 KNL device, green lines are for 2 KNL devices, blue lines are for 3 KNL devices, and black lines are for 4 KNL devices.

The data layout experiments demonstrate that the code is optimized for single device operation, but that if the network communications were fully hidden and the synchronization delays reduced/eliminated, the estimated ideal performance shows minimal difference between single and multiple device compute performance. This result shows that for a given memory layout, and overall working memory level (MCDRAM vs DRAM), the compute time per grid cell per time step is roughly constant. There is an unexpected ‘bump’ in the graphs at a y-bin size of four, that consistently occurs when working in the MCDRAM (does not seem to occur for the problem when it requires significant use of main memory). This bump is unexplained currently, and will be an area of further study. Based on these results, the optimized tile width is determined to be 32 cells wide, which coincided with each tile occupying one 4kB cache page.

4.2 Roofline Analysis

Roofline analysis results show that for the problem sizes investigated here, the compute kernel is primarily bound by the L2 cache bandwidth. The algorithm consistently achieved an arithmetic intensity (operations per byte of data loaded from the L1 cache to core registers) of approximately 0.1. At this arithmetic intensity, the L2 cache bandwidth limits the FLOP rate to approximately 200 GFLOPS Further improvements in the solution would require redesign of the code to increase the arithmetic intensity, or major data restructuring to fit a fundamental work unit/task into L1. The Roofline of the performance of the kernel for various cases is shown in Figure 8.

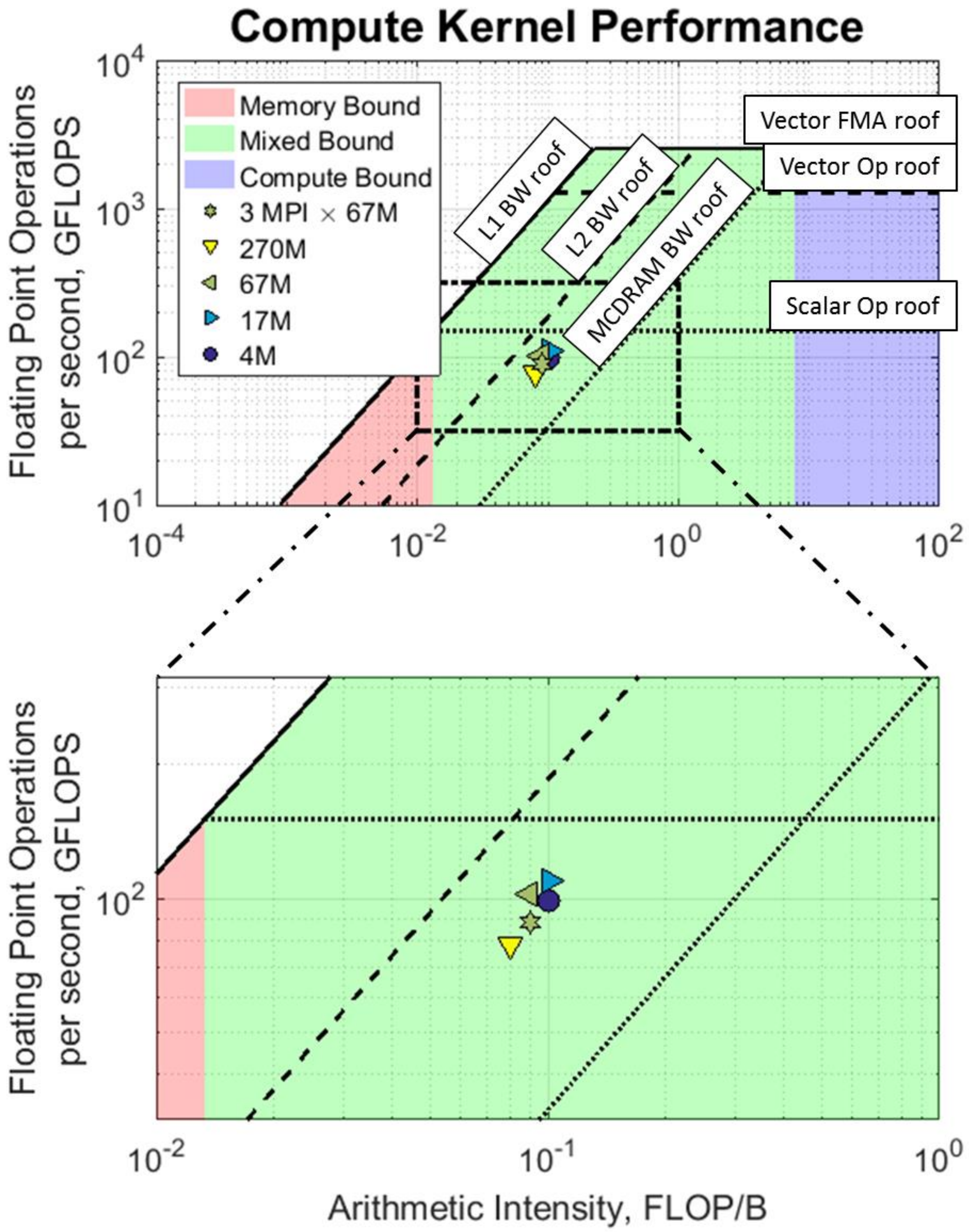


Figure 8: Roofline indicating kernel performance for various cases

Because the roofline analysis shows aggregate performance near (but below) the L2 bandwidth roof, this indicates reasonable optimization, however opportunities to operate more out of the L1 cache will likely increase speedup. Improvement of the use of the L1 cache would enable the code to approach the vector

OP roof of approximately 1.3 TFLOPS, assuming no algorithmic increase in arithmetic intensity. Further algorithmic development to increase the arithmetic intensity would allow further speedup by reducing the rate at which the data is required to be loaded and evicted from higher levels of cache. The kernel shows fairly consistent performance from problem size to problem size, but also shows degraded FLOPS for very large (>67M grid cells) problem sets. The MPI performance demonstrates worse performance from equivalent problem set sizes for single device operation. The algorithm as a whole operates within a region bound by both memory accesses and compute roofs.

Table 1 shows the data that can be used to quantitatively assess the limitations of the compute kernel shown on the roofline plot. The parameters shown highlight compute utilization and memory level accesses. A consideration of instruction mix (scalar vs. vector operations) and pipeline vacancy/instruction retirement provides an assessment on whether the backend is stalled due to operations not completing fast enough or if the core is waiting on data from "deeper" levels of cache/memory to perform the operation. This behavior can be inferred from the data presented in the table.

Table 1: Core back-end performance and limiting factors.

	Compute Utilization		**Memory Accesses**			
Case:	**VPU utilization**	**Backend Bound**	**L2 hit rate**	**L2 hit bound**	**L2 miss bound**	**MCDRAM hit rate**
$4M$	1	0.49	0.937	0.367	0.332	1
$17M$	0.996	0.506	0.936	0.408	0.378	1
$67M$	0.996	0.522	0.933	0.4	0.392	0.928
$270M$	0.996	0.661	0.926	0.189	0.206	0.463
$3MPI \times 67M$	0.996	0.477	0.935	0.332	0.311	0.868

For most cases shown in the Table 1, the kernel has approximately 50% of its execution pipeline empty due to issues arising in the backend. The backend of the processor loads the data for operations and dispatches them to the execution pipeline. It can stall either due to having too many of one type of operation, resulting in unused execution ports of other types of operations, or when data is unavailable to dispatch the operation. The kernel has a good mix of both floating point and integer/address/bit operations along with a very strong utilization (>99%) of the vector processing unit means that the operations in the back-end are unlikely to be the source of the stall, therefore it is most likely a data-supply issue.

For most cases, the kernel profile shows approximately 40% of clock ticks exhibit some pipeline stall per processor due to L2 cache hit handling. Since approximately 93% of L1 misses hit in the L2 for all cases, this would indicate that the problem execution is primarily bound by L2 cache access, which points to the kernel operating as a bandwidth limited code. Of the remaining 7% of L1 misses, approximately 40% clock ticks exhibit some pipeline stall due to L2 miss handling. For most of these cases, the access is found in the MCDRAM (configured to operate as an LLC). These accesses bring the kernel operation further away from the L2 cache bandwidth ceiling. For large problems, however, access to the DRAM is observed. For the 270M case, more than half of L2 misses find the data in DRAM. With its limited bandwidth, this greatly skews the number of back-end bound pipelines to 66.1%. L2 still hits at 93%, but the total increase in L2 miss count leads to a much larger increase in L2 miss handling time. Since the DRAM bandwidth is saturated for this case, The L2 miss bound percent inadequately represents the time due to it being calculated in a fashion that only considers latency, and not bandwidth, thus failing to account for the differing bandwidths of the DRAM and the MCDRAM [28].

4.3 Weak Scaling Analysis

Weak scaling studies were performed to determine code performance for fixed problem size per device. This is useful for determining if larger (in terms of cell count) overall simulations can be solved in similar amounts of time if more compute nodes are used. The weak scaling for this algorithm performs best for problem sizes of intermediate size, near 67M cells per KNL device. Larger problem sets exhibit slowdown associated with excessive MCDRAM/DRAM access, and small problem sets exhibit slowdown associated with reduced amortization of thread and MPI synchronization time. Weak scaling efficiency for the EFIT code on KNL is shown in Figure 9. Performance is generally good for cell counts of 16M-128M per KNL device. Outside of this band, performance starts to degrade due to poor amortization/hiding of communication costs for small problem sizes, and further increased pressure on main memory bandwidth due to more required communication on an already saturated bus, since MPI requires DRAM access as well as the main compute loops.

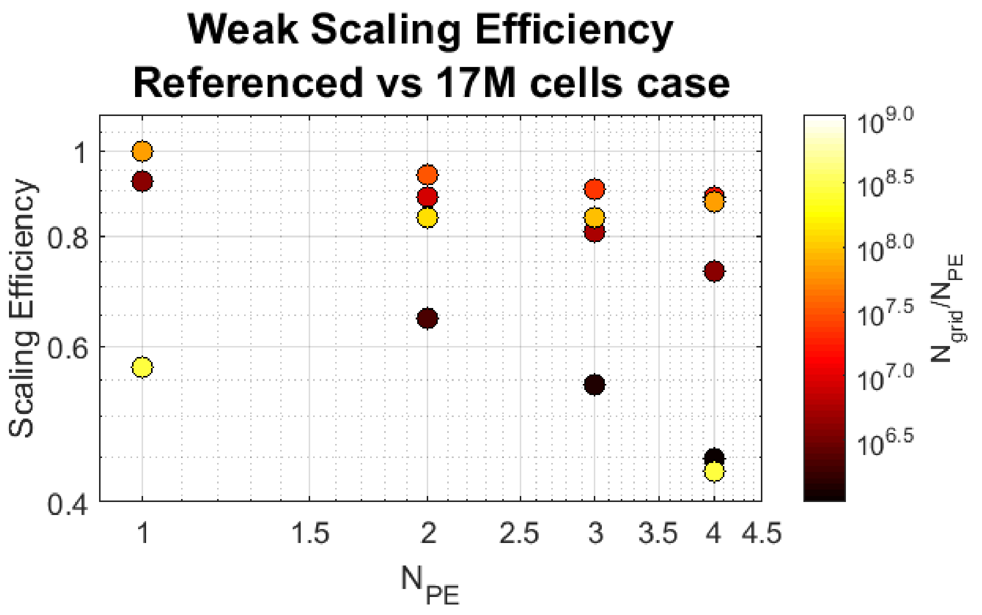


Figure 9: Weak scaling efficiency. Small problem sets have insufficient amortization of communication overhead and large problem sets require accesses to larger, slower memory. The weak scaling efficiency was normalized to the timing of a single node 17M cell case.

This weak scaling analysis shows that for moderately sized (17M-67M cells/KNL) problems, the scaling efficiency stays near unity. This indicates that implementing larger simulations can be achieved by adding a proportionally larger number of KNL devices, allowing the larger simulation to be evaluated in approximately the same runtime. Larger simulation sizes per KNL exhibit slowdowns associated with excessive system memory accesses, and smaller simulations sizes exhibit slowdown associated with reduced amortization of thread and MPI synchronization time.

4.4 Strong Scaling Analysis

Strong scaling tests were also performed, allowing the determination of how the code performs for a given case run on an increased number of KNL devices. Linear strong scaling is shown for problems where the working data set per processor is larger than approximately 4M, but smaller than 270M, as the 270M case is shown to require use of the 'far' DRAM. The strong scaling performance is shown in Figure 10.

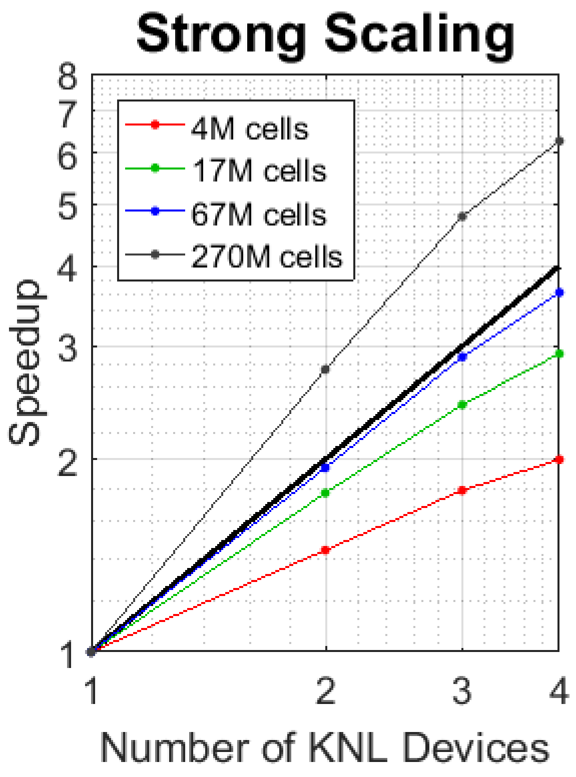


Figure 10: Strong scaling performance. Superscalar performance gains are found for very large working sets, worse scaling performance is found for very small working sets.

Strong scaling analysis shows scalar (speedup proportional to the number of added KNL processors) performance for problem sizes larger than about 4M cells per KNL device, with initially superscalar improvement for the 270M cell case. Strong scaling is maintained until the on-chip calculation size becomes too small, and communications overhead is no longer efficiently amortized over the working set. Very large problems exhibit superscalar behavior when the simulation moves from being worked out of DRAM to being worked out of the MCDRAM (configured as an LLC) and L2 caches. This indicates a performance 'sweet-spot' of problem sizes between 4M cells and 67M. If the whole problem is streaming out of the DRAM with a very reliable access pattern allowing complete latency hiding, then the maximum possible FLOPS would be the arithmetic intensity (≈0.1 FLOP/B) times the bandwidth (≈90 GB/s), which is approximately 9 GFLOPS. The DRAM is still well buffered by the hardware prefetcher keeping the problem working mostly out of L2/MCDRAM, but with 270M grid cells, the DRAM bandwidth begins to overwhelm the prefetcher, and is starts to retard the solution progression.

5.0 Validation

Prior to running an example case, a basic code validation was performed. For verification of the mathematics performed by the optimized EFIT code, a comparison was performed between simulation results and theory. A simulation of a pristine aluminum plate of size 793mm×793mm×6.20mm (2048×2048×16 cells) was run to 298µs (8192 time steps) in time. Guided wave dispersion curves can be created using commercial software tools such as Disperse [29]. Dispersion curves determine the relationship between the guided wave group and phase velocities versus frequency-thickness (frequency of the excitation multiplied by the plate thickness). The frequency-thickness for the simulated case is 1.239 MHz×mm. At this frequency-thickness only two guided wave modes exist, a symmetric (extensional) guided wave mode, S0, and an antisymmetric (flexural) mode, A0. The figure shows the measured value of the A0 mode from the simulation data. It is noted that the A0 mode has a larger amplitude in the out-of-plane direction. Since v_z (out-of-plane) motion was output from the simulation, only the A0 mode group velocity was measured for this comparison.

The group velocity was computed by tracking the movement of the maximum of the Hilbert envelope of the larger-magnitude part of solution. The group velocity at a frequency thickness of 1.239 MHz×mm as determined from the dispersion curve is 3180 m⁄s, and the speed of the tracked Hilbert envelope of the A0 mode in the simulation is 3120 m⁄s. The uncertainty bars were computed based on the standard deviation of the residuals between the linear fit and the position of the maximum of the Hilbert envelope at each returned time step. The group velocity has an error with respect to its dispersion curve of 1.844%.

Phase velocity comparisons for the A0 mode were also performed. Phase velocity is equal to $v_p = f/k$. The A0 and S0 wavenumbers present in the simulation data can be found by first outputting the v_z data on the plate surface at all points in time. Next a 3D FFT is performed and a single frequency slice is selected (corresponding to the excitation frequency). This single slice represents the wavenumber in x and y directions in the simulated plate. As the plate material is isotropic, the wavenumber value is not directionally dependent. The phase velocity for A0 was then evaluated by pulling out the A0 wavenumber and dividing it by the nominal frequency of the drive function. The phase velocity at a frequency thickness of 1.239MHz×mm as evaluated from the dispersion curve is 2460 m⁄s, and the phase velocity as determined from the simulation is 2450 m⁄s. This evaluates to an error with respect to the phase velocity dispersion curve of 0.337%. The uncertainty bars were determined by perturbing the wavenumber by one bin based on the resolution of the 2D FFT.

6.0 Example Case

The optimized EFIT code was tested for an example case of guided ultrasonic waves in an aluminum 2024 plate containing a crack that extends half way through the plate thickness. This case is relevant to NDE detection of damage in aircraft structures. The simulation size was set to 2048×2048×16 cells, with a spatial step size of $\Delta x = 3.87 \times 10^{-4}$m, and time step size of $\Delta t = 3.64 \times 10^{-8}$s. Thus, the total simulate plate size is 793mm×793mm×6.20mm thickness. Guided waves were excited using a 5 cycle 200 kHz frequency Hann windowed sine wave with normal incidence. Figure 11 shows a series of time snap-shots for the out-of-plane (v_z) velocity at the plate surface (i.e., a 2D slice through the 3D simulation space). At later points in time the crack damage leads to scattered waves which, in a sensor array setup, would be detected by a piezoelectric sensor. This crack scatter signal enables damage detection, and in a best-case scenario can also be used for damage sizing. The simulation case shown in Figure 11 was run in 429s, resulting in a time per cell per time step of $0.780\,\frac{ns}{cell \times time\ step}$.

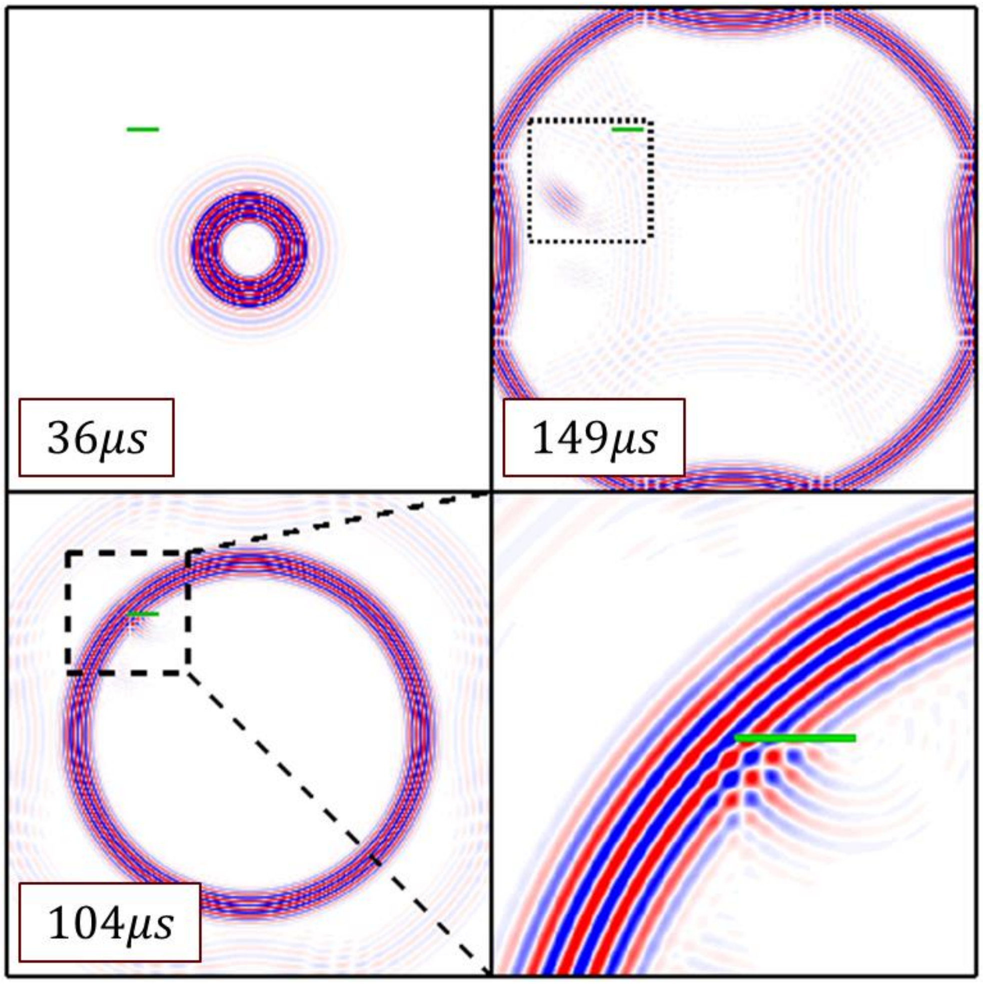


Figure 11: A demonstration of results with a half-thru surface crack. These data were computed with a $\Delta t = 3.64 \times 10^{-8}$s and $\Delta x = 3.87 \times 10^{-4}$m. The model evaluated an aluminum plate 793mm×793mm×6.20mm excited by a 5 cycle 200kHz Hann windowed transducer. The model was run for 298µs. The figure shows time steps at 36µs, 104µs, and 149µs. The reflector is shown in green, and reflected waves are shown in the inset boxes. The lower right depicts a close view of the A0 reflection.

7.0 Conclusions

The optimization of an elastodynamics simulation code for the Knights Landing Many Integrated Core processor was performed, focused on data locality and vectorization. Tiling the data to exploit the cache behavior allowed for significant utilization of the KNL hardware which was demonstrated by carefully profiling code performance using Intel's VTune and Advisor tools. These tools gave low-level data to analyze the KNL resource utilization, allowing targeted optimizations on the highest return options. The MPI implementation allows for a scalable implementation enabling large problems to be simulated.

The model results were validated against theoretical dispersion curves to within 2% of the group velocity, and within 0.5% of the phase velocity of the A0 mode. The validated model performed a simulation of a realistic aluminum plate with a crack which ran on a single device in less than 10 minutes, allowing the ‘slow’ A0 mode to propagate to the domain edge and back to the center. With modest improvements, large ensembles of calculations become possible to aid in characterizing inspections.

Further development will focus on allowing for more modular construction of the simulation domain, while still maintaining or improving simulation speed. Further improvements in larger-scale memory layouts with the possible use of space-filling curves may allow further reduction in bandwidth pressure on the 2D mesh interconnect. Further, improvement of MPI communication hiding and reducing process synchronization would further improve scalability. Overall, the KNL architecture was demonstrated to be capable of rapid and accurate EFIT simulation.

References

[1] N. C. Nguyen, J. Peraire and B. Cockburn, "High-order Implicit Hybridizable Discontinuous Galerkin Methods for Acoustics and Elastodynamics," *Journal of Computational Physics,* vol. 230, pp. 3695-3718, 2011.

[2] G. V. Nivarti, M. M. Salehi and W. K. Bushe, "A Mesh Partitioning Algorithm for Preserving Spatial Locality in Arbitrary Geometries," *Journal of Computational Physics,* vol. 281, pp. 352-364, 2015.

[3] D. Göddeke, D. Komatitsch, M. Geveler, D. Ribbrock, N. Rajovic and A. Ramirez, "Energy Efficiency vs. Performance of the Numerical Solution of PDEs: An Application Study on a Low-power ARM-based Cluster," *Journal of Computational Physics,* vol. 237, pp. 132-150, 2013.

[4] N. Frontiere, C. D. Raskin and J. M. Owen, "CRKSPH - A Conservative Reproducing Kernel Smoothed Particle Hydrodynamics Scheme," *Journal of Computational Physics,* vol. 332, pp. 160-209, 2017.

[5] K. Germaschewski, W. Fox, S. Abbott, N. Ahmadi, K. Maynard, L. Wang, H. Ruhl and A. Bhattacharjee, "The Plasma Simulation Code: A Modern Particle-in-Cell Code with Patch-based Load-balancing," *Journal of Computational Physics,* vol. 318, pp. 305-326, 2016.

[6] J. P. Briggs, S. J. Pennycook, J. R. Fergusson, J. Jäykkä and E. P. Shellard, "Separable Projection Integrals for Higher-order Correlators of the Cosmic Microwave Sky: Acceleration by Factors Exceeding 100," *Journal of Computational Physics,* vol. 210, pp. 285-300, 2016.

[7] Intel, "Products formerly Knights Landing," Intel, [Online]. Available: https://ark.intel.com/products/codename/48999/Knights-Landing. [Accessed 2 February 2018].

[8] A. Sodani, R. Gramunt, J. Corbal, H. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, Y. Liu and Intel, "Knights Landing: Second-generation Intel Xeon Phi Product," *IEEE Micro,* vol. 36, no. 2, pp. 34-46, 2016.

[9] G. Chrysos and Intel, "Intel(R) Xeon Phi(TM) Coprocessor (codename Knights Corner)," in *Hot Chips*, Cupertino, CA, 2012.

[10] C. A. Leckey, M. D. Rogge, C. A. Miller and M. K. Hinders, "Multiple-mode Lamb Wave Scattering Simulations Using 3D Elastodynamic Finite INtegration Technique," *Ultrasonics,* no. 52, pp. 193-207, 2012.

[11] J. L. Rose, Ultrasonic Waves in Solid Media, Cambridge University Press, 1999.

[12] R. M. Levine and J. E. Michaels, "Model-based Imaging of Damage with Lamb Waves via Sparse Reconstruction," *The Journal of the Acoustical Society of America,* vol. 3, no. 133, pp. 1525-1534, 2013.

[13] H. Jiaze and Y. Fuh-Gwo, "Lamb Wave-based Subwavelength Damage Imaging Using the DORT-MUSIC Technique in Metallic Plates," *Structural Health Monitoring,* vol. 1, no. 15, pp. 65-80, 2016.

[14] Z. Tian, C. A. Leckey and L. Yu, "Multi-site Delamination Detection and Quantification in Composites Through Guided Wave Based Global-local Sensing," in *AIP Conference Proceedings*, 2017.

[15] W. H. Ong, N. Rajic, W. K. Chiu and C. Rosalie, "Lamb Wave-based Detection of a Controlled Disbond in a Lap Joint," *Structural Health Monitoring,* 2017.

[16] L. Yu and C. A. Leckey, "Lamb Wave-based Quantitative Crack Detection Using a Focusing Array Algorithm," *Journal of Intelligent Material Systems and Structures,* vol. 9, no. 24, pp. 1138-1152, 2013.

[17] X. Yu, M. Ratassepp and Z. Fan, "Damage Detection in Quasi-isotropic Composite Bends Using Ultrasonic Feature Guided Waves," *Composites Science and Technology,* no. 141, pp. 120-129, 2017.

[18] F. Fellinger and K. J. Langenberg, "Numerical Techniques for Elastic Wave Propagation and Scattering," in *IUTAM Symposium on Elastic Wave Propagation and Ultrasonic Evaluation*, Boulder, CO., 1990.

[19] P. Fellinger, R. Marklein, K. J. Langenberg and S. Klaholz, "Numerical Modeling of Elastic Wave Propagation and Scattering with EFIT - Elastodynamic Finite Integration Technique," *Wave Motion,* vol. 21, no. 1, pp. 47-66, 1995.

[20] R. Marklein, "The Finite Integration Technique as a General Tool to Compute Acoustic, Electromagnetic, Elastodynamic, and Coupled Wave Fields," *Rev. Radio Sci.: 1999-2002,* pp. 201-244, 2002.

[21] J. P. Bingham and M. K. Hinders, "3D Elastodynamic Finite Integration Technique Simulation of Guided Waves in Extended Built-up Structures Containing Flaws," *Journal of Computational Acoustics,* no. 18, pp. 165-192, 2010.

[22] J. P. Bingham and M. K. Hinders, "Lamb Wave Characterization of Corrosion-thinning in Aircraft Stringers: Experiment and Three-dimensional Simulation," *The Journal of the Acoustical Society of America,* no. 126, pp. 103-113, 2009.

[23] C. A. Leckey, K. R. Wheeler, V. N. Hafiychuk, H. Hafiychuk and D. A. Timuçin, "Simulation of Guided-wave Ultrasound Propagation in Composite Laminates: Benchmark Comparisons of Numerical Codes and Experiment," *Ultrasonics,* no. 84, pp. 187-200, 2018.

[24] U. Iturrará-Viveros and M. Molero-Armenta, "GPU computing with OpenCL to model 2D elastic wave propagation: exploring memory usage," *Computational Science & Discovery,* vol. 8, 2015.

[25] P. Huthwaite, "Accelerated Finite Element Elastodynamic Simulations Using the GPU," *Journal of Computational Physics,* vol. 257, pp. 687-707, 2014.

[26] M. Castro, E. Francequini, F. Dupros, H. Aochi, P. O. Navaux and J. Méhaut, "Seismic Wave Propagation Simulations on Low-power and Performance-centric Manycores," *Parallel Computing,* vol. 54, pp. 108-120, 2016.

[27] Intel, Intel(R) 64 and IA-32 Architectures Optimization Reference Manual, Intel, 2016.

[28] Intel, "Intel(R) VTune(tm) Amplifier 2018 Help," [Online]. Available: https://software.intel.com/en-us/vtune-amplifier-help-l2-miss-bound. [Accessed 5 February 2018].

[29] B. N. Pavlakovic and M. J. Lowe, "Disperse: A General Purpose Program for Creating Dispersion Curves," *Rev. Progress of Quantitative Nondestructive Evaluation,* vol. 16, no. A, pp. 155-192, 1997.