Accepted Manuscript

Unstructured Computational Aerodynamics on Many Integrated Core Architecture

Mohammed A. Al Farhan, Dinesh K. Kaushik, David E. Keyes

PII: S0167-8191(16)30056-4
DOI: [10.1016/j.parco.2016.06.001](https://doi.org/10.1016/j.parco.2016.06.001)
Reference: PARCO 2337

To appear in: *Parallel Computing*

Received date: 21 April 2015
Revised date: 28 May 2016
Accepted date: 8 June 2016

Please cite this article as: Mohammed A. Al Farhan, Dinesh K. Kaushik, David E. Keyes, Unstructured Computational Aerodynamics on Many Integrated Core Architecture, *Parallel Computing* (2016), doi: [10.1016/j.parco.2016.06.001](https://doi.org/10.1016/j.parco.2016.06.001)

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Highlights

- Porting implicit unstructured mesh code onto Intel Xeon Phi “Knights Corner”
- Employing aggressive optimizations to improve the CFD flux kernel on Phi hardware
- Exploring different thread affinity modes with different programming paradigms
- Achieving 3.8x speedup using the offload mode compare to the baseline
- Achieving 5x speedup using the native mode compare to the baseline

Unstructured Computational Aerodynamics on Many Integrated Core Architecture

Mohammed A. Al Farhan*, Dinesh K. Kaushik**, David E. Keyes*

** Extreme Computing Research Center
King Abdullah University of Science and Technology
Thuwal 23955-6900, Saudi Arabia*

*** Qatar Environment and Energy Research Institute
Qatar Foundation
PO Box 5825, Doha, Qatar*

Abstract

Shared memory parallelization of the flux kernel of PETSc-FUN3D, an unstructured tetrahedral mesh Euler flow code previously studied for distributed memory and multi-core shared memory, is evaluated on up to 61 cores per node and up to 4 threads per core. We explore several thread-level optimizations to improve flux kernel performance on the state-of-the-art many integrated core (MIC) Intel processor Xeon Phi “Knights Corner,” with a focus on strong thread scaling. While the linear algebraic kernel is bottlenecked by memory bandwidth for even modest numbers of cores sharing a common memory, the flux kernel, which arises in the control volume discretization of the conservation law residuals and in the formation of the preconditioner for the Jacobian by finite-differencing the conservation law residuals, is compute-intensive and is known to exploit effectively contemporary multi-core hardware. We extend study of the performance of the flux kernel to the Xeon Phi in three thread affinity modes, namely scatter, compact, and balanced, in both offload and native mode, with and without various code optimizations to improve alignment and reduce cache coherency penalties. Relative to baseline “out-of-the-box” optimized compilation, code restructuring optimizations provide about 3.8x speedup using the offload

Email addresses: `mohammed.farhan@kaust.edu.sa` (Mohammed A. Al Farhan*), `dkaushik@qf.org.qa` (Dinesh K. Kaushik**), `david.keyes@kaust.edu.sa` (David E. Keyes*)

mode and about 5x speedup using the native mode. Even with these gains for the flux kernel, with respect to execution time the MIC simply achieves par with optimized compilation on a contemporary multi-core Intel CPU, the 16-core Sandy Bridge E5 2670. Nevertheless, the optimizations employed to reduce the data motion and cache coherency protocol penalties of the MIC are expected to be of value for CFD and many other unstructured applications as many-core architecture evolves. We explore large-scale distributed-shared memory performance on the Cray XC40 supercomputer, to demonstrate that optimizations employed on Phi hybridize to this context, where each of thousands of nodes are comprised of two sockets of Intel Xeon Haswell CPUs with 32 cores per node.

Keywords:

Intel Xeon Phi coprocessor, native mode, offload mode, PETSc-FUN3D, thread affinity, unstructured meshes

1. Introduction

The shared-memory nodes that make up contemporary distributed-memory systems employed for computational fluid dynamics are evolving from “multi-core” to “many-core,” where the distinction is not simply in the number of cores, but in their complexity. “Many-core” processors, such as Intel’s Xeon Phi, exploit the concurrency still available from Moore’s Law while targeting reduced power consumption per flop with simpler, slower cores that lack some of the latency-tolerant features of the powerful cores of “multi-core” processors, such as Intel’s Sandy Bridge. While some kernels quickly saturate in performance with the increasing number of cores sharing a single hierarchical local-memory system, others that are compute-bound on traditional CPUs can benefit from the emerging architectures, with due attention to data placement. Partial Differential Equation (PDE)-based simulations typically possess such compute-bound kernels when sophisticated discretizations are employed on problems with multiple components, allowing significant reuse of arguments in registers and caches. However, many-core architectures must deal with increased intra-node data sharing. Due to the irregular ordering of reads and writes in an unstructured-mesh PDE-based application, it is nontrivial to obtain high performance in such a hierarchical memory environment. We evaluate herein, on the Intel Many Integrated Core (MIC) architecture, one such kernel from PETSc-FUN3D, an unstructured tetrahe-

dral mesh Euler flow code that has been optimized previously for high performance in distributed memory scaling [1, 2] and multi-core shared memory scaling [3, 4].

Intel’s Xeon Phi many-core accelerator technology has become pervasive enough to feature in some globally top-ranked supercomputer systems, including TianHe-2, which heads the Top500 list at the time of writing. It has been adopted for the highest ranking supercomputers for open science of the U.S. NSF (Stampede, 2013) and DOE (Cori, 2016), in the “Knights Corner” and “Knights Landing” versions, successively, with “Knights Hill” technology contracted for deployment in DOE’s Aurora system (expected in 2018). This has focused programmer interest on many-core technology, especially since such accelerators have evolved to independent compute nodes, able to operate without reliance on a conventional CPU. To complement strides in hardware performance, hybrid-programming paradigms [5, 6] must evolve to effectively utilize exascale computers, which may consist of a million nodes of a thousand cores each [7, 8]. In that context, a message-passing programming model may co-exist with a shared-memory programming model as a means to effectively exploit all degrees of hardware concurrency. For PDE-based codes, weak scaling using the message passing library MPI is a proven technology. Shared-memory paradigms such as OpenMP are less far proven with respect to many-core implementations, which motivates the current study.

Large-scale parallel simulations of Euler and Navier-Stokes flows based upon local finite discretizations (for unstructured meshes, typically finite volumes) on quasi-static meshes weak scale well on distributed memory architectures through carefully load-balanced domain decomposition. However, their performance as a percentage of peak is poor without careful attention to data layout and access ordering. While implicit PDE-based codes are generally plagued with memory bandwidth bottlenecks due to lack of reuse of cached Jacobian matrix elements in the solver phase, PDEs solved on unstructured meshes pose even greater challenges due to indirect addressing, which further dilutes the fraction of floating point operations among all operations. It is therefore of interest to study the execution amenability of unstructured PDE-based codes on state-of-the-art many-core architectures. A system that has a suitable hardware balance for one phase of the computation may be strongly out of balance for another, and careful phase-by-phase profiling is required to know where and how to invest code optimization effort [9]. Insights gained can be used to fine-tune the codes, using all available algorithmic freedom to seek optimal performance, and to provide feedback

to influence future architectures.

This paper is structured as follows. Section 2 reviews the background of the compute-intensive flux kernel of the incompressible Euler version of PETSc-FUN3D code, a fully nonlinearly implicit unstructured mesh code with joint origins at NASA and the U.S. DOE [1]. Section 3 recalls key performance characteristics of this and similar PDE-based codes. Section 4 describes the specifications of the MIC hardware employed, with details deferred to Appendix B, and describes the input data sets. In section 5, we highlight the key optimizations procedures through which we improve the performance of the flux kernel on the many-core hardware. In Section 6, we play with several modes of memory mappings and thread mappings and discuss the results. Section 7 concludes with a summary and prospects for future work.

2. Background

Fully Unstructured Navier-Stokes in 3 Dimensions (FUN3D) is a tetrahedral, vertex-centered, unstructured mesh research code written in Fortran for solving Navier-Stokes and Euler equations (see Appendix A) of fluid flow in incompressible and compressible forms. It was originally developed under the direction of W. Kyle Anderson at NASA Langley Research Center (LaRC) [10, 11] in the early 1990s and has become a widely used resource [12], ported at its current extremes to nearly 100,000 cores and supporting meshes of up to one billion vertices [13]. FUN3D employs control volume discretization with variable-order (second-order accurate in space) Roe schemes for approximating the convective fluxes, and a Galerkin-type diffusion discretization for the viscous terms. It has been used in a number of complex large-scale problems including performance prediction and design optimization of airplanes, automobiles, and submarines. An artificially coarse example of a FUN3D domain exterior to a three-dimensional ONERA M6 wing is shown in Figure 1; where the wing surface triangulation is shown in green; the symmetry root plane in red; and the far-field boundary in blue.

The Euler subset of FUN3D was forked for study of distributed memory scaling in 1998 and restructured to employ the Portable, Extensible Toolkit for Scientific computation (PETSc) solver toolkit [14, 15, 16]. PETSc-FUN3D performance is thoroughly discussed, analyzed, and modeled in [1], which culminated in the 1999 Gordon Bell Special Prize that ran on the world's then most powerful supercomputer, the Intel ASCI Red machine at Sandia

Figure 1: The surface triangulation of the ONERA M6 wing

[2]. PETSc-FUN3D has since been used as representative of many unstructured mesh PDE-based applications for HPC architectures. For instance, Bhowmick *et al.* developed an adaptive, polylogarithmic approach based on machine learning heuristics to automatically select the linear solve method at each nonlinear iteration of the compressible subset of PETSc-FUN3D [17]. The adaptive procedure dynamically shifts between different Krylov subspace methods, based upon the numerical characteristics of the linear systems obtained throughout the route of the nonlinear iterations, to overcome the hurdles arising in the steps of numerical solution.

In [3], the incompressible version of PETSc-FUN3D was revisited to investigate the modest thread-level parallelism available on the IBM Blue Gene/P architecture. Several data partitioning schemes involving OpenMP threads were investigated to improve its performance. Furthermore, in [4], a team led by Intel successfully explored several shared-memory optimization techniques on the incompressible subset of PETSc-FUN3D running on a contemporary Intel Sandy Bridge multi-core processor. These most recent studies show the potential for hybrid-programming paradigm compared to pure MPI, with careful consideration of the temporal and spatial locality of the data close to the CPU, work division among processes and threads, and update management of ghost points. In particular, a speedup of 6.9x is achieved for the

entire PETSc-FUN3D application on 20 threads (2 threads each on 10 cores) in [4], relative to out-of-box baseline compilation. While the triangular solve phase of the ILU preconditioner in the Newton-Krylov solver obtains only 3.2x speedup, the flux kernel achieves a full 20x speedup, with other major kernels performing between these two extremes.

3. PDE-based Computational Kernels

We relegate most details of the fluid dynamics model, discretization, and solver to previous publications [1, 2, 3, 4, 17] for reasons of space and focus here on transformations that address the new hardware context. In a typical implicit mesh-based PDE solver, there are four readily identifiable and distinct groups of computational tasks that stress different hardware subsystems of a high performance computer [18]. In the context of a vertex-centered control volume discretization, like FUN3D, in which the data are cached at the edge vertices, these are:

1. Vertex-based loops
 - Update auxiliary (e.g., geometry and constitutive parameters) and state (e.g., momenta and pressure) vector data
2. Edge-based “Stencil Operations” loops
 - Residual vector and Jacobian matrix evaluation
 - Jacobian-vector products (often replaced with matrix-free form, reverting to residual evaluation)
 - Interpolation between mesh levels in multilevel solvers
3. Sparse, narrow-band recurrences
 - Approximate factorization
 - Back substitution
 - Relaxation/smoothing
4. Global reductions (vector inner product and norms)
 - Orthogonalization/conjugation
 - Convergence progress checks
 - Stability heuristics

The performance bottlenecks of a PDE-based code can usefully be approached with simple models in which each critical computational resource is in turn assumed to be the limiting one, assuming other resources to be present in excess. Edge-based loops typically consume the greatest share of execution time, except when excessive strong distributed memory scaling leads to subdomains in which surface vertices dominate enclosed volume vertices, which limit should be avoided. The arithmetically heaviest edge-based loop is the the flux kernel, which traverses all vertices during the Newton residual evaluation, the application of Jacobian-free matrix-vector products for the Krylov method, and the formation of the preconditioner for the Jacobian matrix [1, 4]. In FUN3D, this kernel calculates the conservation law residuals in Voronoi cells whose irregular faces bisect the edges between each pair of adjacent vertices. It reads the flow variables that are stored at the vertices at each iteration. The iteration traverses an entire vertex-based array in a strategically chosen order to read state variables, compute fluxes, and update residual values at every vertex; see Figure 2 and Algorithm 1. This kernel is our focus, since its performance should be an upper bound on the overall performance of PETSc-FUN3D on a many-core accelerator designed for compute-intensive operations, similarly to what was demonstrated for multi-core in [3, 4].

Algorithm 1 PDE Flux Kernel (an Edge-based Loop)

- 1: **for each** $e \in \text{array of edges}$ **do**
 - 2: Get the index of the right node of e
 - 3: Get the index of the left node of e
 - 4: Read the flow variables from both endpoints of e
 - 5: Compute the flux and the residual
 - 6: Perform write-back operations to update the flow variables of both endpoints of e
 - 7: **end for**
-

3.1. Pseudo-Transient Newton-Krylov-Schwarz (Ψ NKS) Euler Flow Solver

The Newton-Krylov-Schwarz (NKS) solver of PETSc is a synergistic combination of a Newton-Krylov nonlinear method and a Krylov-Schwarz preconditioned linear iterative method for a general algebraic problem [19, 20]. These equations are discretized as $f(u) = 0$, where u is a fully coupled

Figure 2: Tetrahedral mesh PDE flux kernel

vector of unknowns of the system state, and $f(u)$ is the vector-valued function of the residual of nonlinear conservation laws resulting from the spatial discretization. In this application, $f(u)$ derives from a time-implicit control-volume discretization of the Euler equations with artificial compressibility, with pseudo-transient continuation used to accelerate convergence to a steady state. The original FUN3D has additional full compressibility and Navier-Stokes options, which when activated increase the prominence of the flux kernel. They also increase the opportunities for spatial and temporal locality and therefore for enhancing arithmetic intensity; we deal here with the most austere case of incompressible Euler; refer to Appendix A for a summary.

The Jacobian-free method implemented in PETSc avoids explicitly storing a full, consistent Jacobian matrix, by relying only on sparse Jacobian-vector multiplications to access the coefficient matrix. However, in practice, subdomain-sized diagonal blocks of a preconditioner matrix should be computed to keep the number of Krylov iterations manageable. These preconditioners

tioner blocks may employ a lower-order discretization than the true Jacobian. For a complex discretization involving flux-limited Euler they are typically computed by finite differences of residual evaluations on graphs, suitably covered to minimize the number of residual evaluations required. Newton's method can be globalized by pseudo-transient continuation (Ψ tc) [21, 22].

The four nested levels of Ψ NKS implicit framework are shown in Algorithm 2. The flux kernel is called in the steps numbered (2), (4), and (9). Global inner products, which expose any load imbalance, are performed in steps (5), (7), and (9). Steps (3) and (8) are typically memory bandwidth-bottlenecked.

Algorithm 2 Ψ NKS algorithm

```

1: for  $i = 0$  to  $max\_timestep$  do
2:   compute pseudo-timestep (1)
3:   for  $j = 0$  to  $max\_Newton$  do
4:     compute nonlinear residual and Jacobian (2)
5:     for  $k = 0$  to  $max\_Krylov$  do
6:       precondition subdomain problems concurrently (3)
7:       perform Jacobian-vector product (4)
8:       enforce Krylov basis conditions (5)
9:       update optimal coefficients (6)
10:      check linear convergence (7)
11:    end for
12:    perform DAXPY update (8)
13:    check nonlinear convergence (9)
14:  end for
15:  compute the aerodynamic forces and moments (10)
16: end for

```

4. System and Input Deck Specifications

Three sets of experiments are described herein. The most novel are variations of a many-core implementation. We compare these with a multi-core implementation of the same kernel. We also perform multi-node, multi-core experiments on a leading recently Cray XC40, adding the XC40 to a growing list of supercomputers on which FUN3D has been benchmarked, stretching back more than 15 years.

For the many-core experiments, we use a Linux server equipped with dual Intel Xeon Phi 7120P “Knights Corner” coprocessors. The Phi cards are hosted by a dual socket Sandy Bridge E5 – 2670 CPU. Each socket consists of 8 hardware cores (in total 16 cores) that can run up to two threads per core by enabling Intel hyper-threading technology. Each CPU core has two 32KB L1 caches, for data and for instructions. In addition, each core has one 256KB L2 cache. All cores share an L3 cache with size of 20MB. The CPU clock speed is 2.6GHz. Both sockets share a 65GB main memory.

For the multi-core experiments, we use a Linux server that has two sockets of the Intel Haswell E5 – 2699V3 CPU. Each socket consists of 18 cores (in total 36 cores) that can run up to two threads per core by enabling Intel hyper-threading technology. The size of the L1 and L2 caches are similar to the Sandy Bridge E5 – 2670 CPU, whereas the L3 cache size is 45MB that is shared between all of the cores. The CPU clock speed is 2.3GHz. The server has a 264GB main memory.

For the large-scale experiments, we use Shaheen II at KAUST, which was ranked 9th on the Top500 list of November 2015. The system consists of 6,174 compute nodes, each of which is equipped with a dual socket Intel Haswell E5 – 2698V3 CPU. Each socket contributes 16 cores (in total 32 cores per node; thus, the whole system consists of 196,608 compute cores) that can run up to two threads per core by enabling Intel hyper-threading technology. The size of the L1 and L2 caches are similar to the Sandy Bridge E5 – 2670 CPU as well as Intel Haswell E5 – 2699V3 CPU, whereas the L3 cache size is 40 MB and is shared between all of the cores. The CPU clock speed is 2.3 GHz. Both sockets share a 128 GB main memory and the entire memory system consists of 786 TB. Compute nodes are connected by the Cray Aries interconnect with dragonfly topology, with a maximum of 3 hops for a message between any pair of nodes. Theoretically, Shaheen II has a peak performance of 7.2 Pflop/s.

4.1. Intel Xeon Phi Coprocessor at a Glance

We defer detailed information about Intel Phi architecture to Appendix B. The Intel Phi 7120P “Knights Corner” has a total 61 hardware cores running at a nominal speed of 1.238 GHz, or approximately half the speed of the CPU. The implementation of the micro-architecture (see Figure 3) of the Phi’s cores is based on the x86 Instruction Set Architecture (ISA) with an extended Advanced Vector eXtension (AVX) to support 512-bit wide Vector Processing Unit (VPU), which provides Single Instruction, Multiple Data

(SIMD) instructions [23]. The VPU supports Fused Multiply-Add (FMA) instructions that perform either eight 64-bit values or sixteen 32-bit values of floating point operations per a single CPU cycle. The cores use Intel hyper-threading technology to run up to four hardware threads simultaneously, i.e., the Phi supports four-way Simultaneous Multi-Threading (SMT)¹ [24]. Hence, in total the Xeon Phi chip can run up to 244 thread contexts concurrently.

Figure 3: Intel Xeon Phi coprocessor architecture

Every MIC core has two 32KB L1 coherent caches for data and instructions, respectively, as well as a 512KB L2 cache. The caches are divided into several lines of 64 bytes. All L2 caches are interconnected with each other and the memory controllers via a bidirectional high-bandwidth 512-bit ring bus, namely On-Die Interconnect (ODI); effectively ODI creates a shared last-level cache amounting to 32MB [25]. The Phi chip is connected to a

¹Intel Xeon Phi's SMT can not be disabled, unlike other Intel Xeon architectures that support hyper-threading.

16GB on-board Graphics Double Data Rate, version 5 (GDDR5) Dynamic Random Access Memory (DRAM).

Theoretically, the peak performance of Intel Xeon Phi² is 1,208.29 GFlop/s (double precision floating point) or 2,416.58 GFlop/s (single precision floating).

Three different programming paradigms are available on Xeon Phi. One uses the die as an independent compute node by running the code directly on Xeon Phi as a separate many-core Linux node; this is called “native mode.” The second is a so-called “symmetric” mode that allows the coprocessor to establish communication with other devices through MPI routines, so that the Phi works as an independent compute unit along with the CPU (this mode is not considered herein). Finally, an “offload mode” is offered, similarly to other accelerators like GPUs. In this mode, a portion of the code (normally the most compute-intensive kernels) with its associated input data is rolled onto the Phi [26] and the results rolled back.

With any of these programming paradigms, the Xeon Phi coprocessor provides three modes of thread affinity, named “scatter,” “compact,” and “balanced.” The scatter mode performs the most balanced possible distribution of threads across all cores, spreading threads around with a round-robin placement, so that each thread has the most exclusive access to the core resources. Scatter mode allocates a single core for each thread even if the number of the threads launched is less than 61. The compact mode attempts to use least amount of cores by spawning 4 threads to a core before filling the next one. It maximizes cache utilization while saving power by launching the minimum number of cores, since unassigned cores draw low power [27]. The balanced mode equally distributes threads across all cores such that all threads pinned to the same core have consecutive thread IDs [28].

The thread IDs in the scatter mode are not physically contiguous. Instead, adjacent thread IDs are physically distributed so that they are more likely to be physically nonadjacent. In contrast, the thread IDs in both balanced and compact modes are physically adjacent. In the case of maximum thread count allocation (244 threads), the balanced and compact modes identically have the same thread placements [27]. Figure 4 shows examples of different thread distributions among 60 cores of Xeon Phi; the empty white

²8 Double Precision or 16 Single Precision $\times$ 2 FMA $\times$ 61 cores $\times$ 1.238 GHz Clock Speed.

slots represent the idle thread contexts within a core of each mode of allocation. If all of the four thread contexts within a core are idle (four empty slots), then the core is not being launched in this mode of the allocation.

Figure 4: Thread distribution among Xeon Phi cores (zero-based numbering)

The main advantage of using the Phi over other accelerators is that the programming effort can quickly focus on how to achieve optimal performance, rather than how to port the code, itself. With well-known parallel programming paradigms such as POSIX Threads (pthreads), OpenMP, OpenCL or MPI, developing applications on Phi is easily accomplished. Tuning the code, on the other hand, can be a daunting proposition at present. Furthermore, to exploit the Phi's peak performance, applications must fully utilize all cores and their VPUs by keeping them busy throughout the execution cycle. The data has to be ready at the cores' disposal without delivery delays [29].

4.2. Software Stacks for the PETSc-FUN3D Experiments

We use the Intel ICC compiler (Intel C/C++ v16.0.1) and version 3.5.1 of the Intel Manycore Platform Software Stack (MPSS). We use PETSc version

3.7 built on top of Intel MPI library version 5.1.2 and Intel Math Kernel Library (Intel MKL) Version 11.3.2. For partitioning vertices, we use version 5.1.0 of the MeTiS partitioner [30].

For the executions on Shaheen II, we use the latest versions of the Cray software stacks, as follows: Cray C compiler, Cray PETSc library, Cray BLAS & LAPACK, and Cray MPI library (Cray-MPICH). Also, we use MeTiS 5.1.0 to partition the overall domain for MPI, and the subdomains for shared memory optimization.

For reproducibility, we give the PETSc settings of Ψ NKS algorithmic framework as follows:

- Scalable Nonlinear Equations Solvers (SNES) component: Backtracking (bt) Newton Line Search (SNESLineSearch)
- Maximum number of SNES iterations per pseudo-timestep: 1
- Scalable linear equations solvers (KSP) component: GMRES [31] with 30 restart size
- Maximum number of KSP iterations per Newton step: 60
- Preconditioner (PC) components:
 - Restricted Additive Schwarz, with one subdomain block per process and cell overlap of 1
 - Subdomain solve settings are:
 - * Sub KSP solve is Preonly (apply only the preconditioner)
 - * Sub PC type is “out-of-place” incomplete LU factorization with one level of fill (ILU(1))

The initial CFL³ value for pseudo-timestepping is set to 50.0, with the maximum CFL value (effectively steady-state) set to 1.0e+05. The pseudo-timestep grows adaptively according to a power law function of residual reduction [1]. FUN3D’s artificial compressibility parameter (β) is set to 15.0, and the angle of attack parameter (α) of the ONERA M6 wing is the standard 3.0 degrees with Mach number 0.84.

³Courant–Friedrichs–Lewy condition.

4.3. Input Data Sets

The main objective of this paper is to study and address the challenges of unstructured mesh computations running on a many-core architecture. Thus, a small ONERA M6 wing mesh that consists of 22,677 mesh vertices ($22,677 \times 4 = 90,708$ DOFs⁴) and 146,384 edges across all programming paradigm and affinity options. A small mesh guarantees the spatial and temporal locality of the data elements.

To further understand the performance of the Xeon Phi implementation of PETSc-FUN3D, we have also used a finer mesh on a single node. This finer mesh consists of 357,900 mesh vertices ($357,900 \times 4 = 1,431,600$ DOFs) and 2,438,109 edges.

For large-scale experiments on Shaheen II, we use a yet finer mesh generated for the same geometry that consists of 2,761,774 mesh vertices ($2,761,774 \times 4 = 11,047,096$ DOFs) and 20,418,102 edges.

We do not further address yet larger problem sizes on Intel Phi, except to mention that FUN3D has no intrinsic limits on mesh refinement in weak distributed memory scaling, unless the linear iterations become impractically ill-conditioned, or until integer limitations are reached for labeling entities, or the resolving power of floating point precision is reached for tiny mesh cell dimensions.

5. Shared Memory Optimizations

Since we employ an unstructured mesh, the code performs many integer operations per flop due to indirect addressing and non-unit stride memory access patterns. The irregular access pattern leads to cache misses that squander many cycles. Therefore, using ICC compiler automatic vectorization with the irregular memory access patterns is less efficient, and it considerably degrades the performance of unstructured computations on MIC hardware. In the Phi context, the coprocessor can load from the DRAM 16 consecutive single precision values or 8 consecutive double precision values at every clock cycle with a single vector instruction. This is useful when these values are aligned with each other in the memory, as is typical of structured grid computations [33]. However, in the unstructured mesh context, these values are not generally stored adjacently in the address space; they therefore

⁴Degrees Of Freedom [32].

lead to a noncontiguous memory access pattern. Thus, the coprocessor has to load them with multiple vector instructions, each of which is executed at a different cycle. Listing 1 shows an example of indirection addressing written in C programming language. It is challenging to vectorize computations that include noncontiguous memory access patterns by the compilers alone [34].

```

1 // A, B and I are arrays of size N
2 // I stores indices to reference B
3 // Referencing B using I is called indirect addressing
4 for(i = 0; i < N; i++) A[i] *= B[I[i]];

```

Listing 1: Example of indirect addressing

Indeed, the conventional means for optimizing cache data locality may effectively apply on the MIC hardware. For example, data blocking and field interlacing, which has been implemented on PETSc-FUN3D [1], elevates the performance of our Phi implementation, although we further tune the initial implementation to favor Phi cache design. Nonetheless, these traditional techniques in general are not enough for rewarding performance gains. Hence, we investigate herein additional techniques that have been explored in literature for shared-memory optimizations.

To improve compiler vectorization and data reusability, data structures may be aligned on specific byte boundaries, which can favor MIC cache-line large size. Thus, only a single cache-line access that requires one instruction is performed [28]. Data structure alignment effectively reduces the indirect addressing, especially when a given architecture supports a large enough size of cache-lines. It minimizes memory load latencies, while aiding the compiler to generate a very efficient vector code [35]. Furthermore, data alignment improves the data transfer over the PCIe bus in the offload mode from the CPU to Phi and vice versa. The PETSc-FUN3D data structures were restructured and reallocated with 64-byte alignment using extra contentless bytes for padding, which utilizes the 512-bit wide vector register within a single cycle. Also, memory padding and alignment subjugate the hurdles of set associativity of L1 and L2 caches, when several cache lines are mapped to the same set (i.e., conflict misses). We adjust the PETSc-FUN3D data objects that are related to each other. In so doing, we regulate access patterns of address space by situating the starting addresses of their data structures. Since the flux kernel is a loop over mesh edges, the improvement allows the

kernel to process 8 consecutive edges simultaneously within a single thread using each SIMD lane.

Each OpenMP thread owns private temporary buffers that locally compute their work portions, which are associated with their assigned edges. When a thread finishes its computations, it writes the private temporary buffers at once to update the vertex data (sometime it updates two vertices, its vertex plus the ghost vertex, if the edge is shared between multiple threads (*a cross-edge*)). Private local buffers effectively overcome issues of core-to-core communication, when the requested data is available in neighboring caches. On the other hand, they aid the compiler to use Phi's streaming stores that are provided at the instruction-level, through which they ensure that the memory writes do not require a read operation before the writing. Also, they reduce Translation Lookaside Buffer (TLB)⁵ pressure. Overall, these improvements benefit the Intel ICC compiler auto-vectorization by avoiding code branches (*if* constructs) [37]. Ultimately, the large SIMD processing unit provided by MIC benefits from these optimization techniques that instructs the ICC intrinsic auto-vectorization.

We use Reverse Cuthill McKee [38] to order the vertices. This has been implemented in PETSc-FUN3D [1] for cache-based architectures, and we have further improved this implementation to accommodate Phi's cache architecture. This enhances the spatial locality and reduces cache misses. Reverse Cuthill McKee renumbers the edges to maximize the reuse of vertices, to minimize the penalty of indirect addressing. In addition, we use a "structure of multiple arrays" ("struct-of-arrays") data layout to store the mesh data, which benefits from the MIC VPU gather/scatter pipeline for sparse data that issues multiple sequential loads for each data field to fill the VPU. The array data structures of the mesh information are encapsulated in several C structs, forming a struct-of-arrays storing arrangement of multiple data streams, which improves vectorization and both spatial and temporal locality of reference via supplementing a unit-stride references. Furthermore, the Phi's implementation of gather/scatter instructions enables loading from non-continuous memory addresses to the SIMD register [39]. This extracts the SIMD-level parallelism of the flux kernel, and optimizes the cache footprint while reducing TLB misses. It also improves transfer of the data el-

⁵TLB is a cache layer implemented in the Memory Management Unit (MMU) by the hardware, used to speedup the virtual address to physical address translation [36].

ements and instructions from memory to the L1 and L2 caches (software prefetching to hide memory latency).

Unstructured mesh computations are plagued by irregular data dependency [40]. Data partitioning may carry a large performance penalty, and is an essential consideration for shared-memory parallelism. MeTiS was employed to decompose the subdomain among the OpenMP threads in the original PETSc-FUN3D. This technique was investigated for a modest number of threads available at the time per each MPI rank [3]. MeTiS attempts to partition vertices into subdomains to evenly balance the computational work between the threads, which minimizes the replication of edges across partitions and applies “owner computes”⁶ rule at the thread-level. MeTiS partitions the data so that the owner of the nodal data performs the computation to update its own elements. Although the MeTiS assignment of vertices to threads has been shown to significantly outperform other threading approaches, the edge replication slightly increases as the number of threads increases. Additionally, in [4], the authors conduct early experiments with PETSc-FUN3D using 240 threads running on Phi chip, and the overhead of the edge replication exceeds 15%. Hence, we further extend the implementation of MeTiS to optimize the thread-level parallelism on Phi, and improve the scalability by reducing the replication overhead. In our optimizations, we do not rely only on balancing the computational work between the threads. We also balance the communication between the threads to overcome core-to-core obstacles that are introduced by Phi’s cache-coherency challenges. This reduces the edge-cuts and the total communication volume, by decreasing the amount of the adjacent elements assigned to different working threads [9, 42]. Algorithmically, this also results in performance improvement in the linear system solver since the parallel single-level additive Schwarz preconditioner is stronger with the hybrid case because of the usage of larger subdomains.

6. Performance Results and Discussions

We start with a baseline execution against which our fine-tuning efforts for shared-memory parallelization of the flux kernel can be compared. We develop two models based on the “out-of-box” baseline execution; one for the

⁶The “owner computes” rule is a very common and simple parallel distribution strategy that relies on assigning computations to processors that own the left-hand side data element of the array [41].

offload mode and the other one for the native mode. The baseline models use the original incompressible PETSc-FUN3D code [1, 3], which employs MeTiS to partition the subdomain among the OpenMP threads. In addition, since PETSc-FUN3D is a legacy code, which is compatible only with the older versions of PETSc framework, we first port the baseline code into the latest version of PETSc framework, by updating several calls of PETSc routines and distributed data structures.

6.1. Offload Baseline Model

In this baseline model, the flux kernel is offloaded onto the Intel MIC chip as a coprocessor. At each kernel call, the auxiliary, primary, gradient, and solution vector data structures are copied from the host to the device; whereas the residual vector data structure is copied back from the device memory to the host to keep both memories up-to-date.

Figure 5 shows the offload baseline strong scalability study of the flux kernel running on the Phi with different thread affinities. The initial study compares the running time between different thread placements with respect to the number of the running threads.

The running time of the flux kernel decreases monotonically with the number of OpenMP threads, with scatter affinity mildly superior at intermediate numbers of threads. Also, the balanced affinity type has roughly the same runtime as the compact mode.

As mentioned earlier, scatter mode causes each core to consume power while the resources of each core, from L1 cache to the registers, are exploited by a single thread. For 120 and 240 threads, the execution time is almost the same with the three affinity modes. This is due to the number of running threads being greater than the number of cores. As a result, the advantage of exploiting cores through scatter mode vanishes.

One of the downsides of using the offload mode for the Phi as a coprocessor, or any other accelerator hardware, is the overhead of the data transmission between the host (CPU) memory and the device (coprocessor) memory. The link (PCIe bus) bandwidth limitations⁷ may drastically limit the performance of the application. The data movement in such system can be the “make-or-break” feature of the implementation. We tune the data transmission techniques as follows.

⁷In our experimental system, two PCIe v2.0 connect each MIC chip with the CPU, each of which has 5GT/s transmission speed.

Figure 5: Baseline model: Strong thread scaling of the flux kernel of “out-of-box” baseline compilation of PETSc-FUN3D running on the CPU; the flux kernel is offloaded onto Intel Xeon Phi (22,677 mesh vertices)

The data structure, which is used to store the auxiliary and primary fields, is allocated and transferred from the CPU memory to the Phi memory before the execution of the solver. These data structures are stored and retained in the Phi memory during execution. A once-and-for-all transmission of these data structures saves time that was earlier consumed in each flux kernel. The CPU spends on average about 0.5 seconds out of the bulk execution time carrying out this transmission, which includes Coprocessor Offload Infrastructure (COI) initialization [43, 44]. On the other hand, the gradient vector used in flux limiting, as well as the solution vector, are copied from the CPU memory to the MIC memory at each flux kernel call. The residual vector is copied from the MIC memory to the CPU memory. These data structures are updated continuously during execution of the solver. Our tim-

ings capture only the flux evaluation portions. The memories on both chips must be synchronized by updating them regularly during execution.

6.2. Native Baseline Model

This model compiles the original CPU code with “*mmic*” flag without any further modifications on the physics or solver kernels, and runs the whole PETSc-FUN3D code natively on Intel MIC as an independent Linux compute node.

Figure 6 shows the native baseline strong scalability study of the flux kernel running on the Phi with different thread affinities.

Figure 6: Baseline model: Strong thread scaling of the flux kernel of “out-of-box” baseline compilation of PETSc-FUN3D running natively on Intel Xeon Phi (22,677 mesh vertices)

Similar to the offload code, the running time of the flux kernel decreases monotonically with the number of OpenMP threads. Also, scatter as well as

balanced affinity type outperform the compact mode with different number of threads except when the maximum number of threads is launched (i.e. 240 threads).

6.3. Offload and Native Mode Performance Results With the Coarse Mesh

Figure 7 shows the offload mode improvements of the flux kernel compared to the (best) scatter affinity type of the baseline model. The performance of the three affinity modes is almost the same, since our optimizations aim to maximize both core as well as cache utilizations. The results present about a 3.8x speedup with a single thread and a large number of threads relative to the baseline.

Figure 7: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel offloaded onto Xeon Phi compared to the baseline model (22,677 mesh vertices)

Figure 8 shows the native mode improvements of the flux kernel compared to the (best) scatter affinity type of the baseline model. The results present

about a 5x speedup with a single thread and a large number of threads relative to the baseline.

Figure 8: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel running natively on Xeon Phi compared to the baseline model (22,677 mesh vertices)

To further demonstrate the data transmission overhead, Figure 9 shows about a 1.8x speedup in the performance of the baseline native mode for asymptotic numbers of threads compared to the baseline offload mode. Figure 10 shows about 1.4x speedup in the performance of the fine-tuned native mode for a single thread compared to the fine-tuned offload mode. Also, it shows 2.5x, 3.5x, and 4.7x speedup in the performance for 60, 120 and 240 threads respectively.

To investigate the feasibility of using MeTiS rather than other partitioning techniques, Figure 11 shows the performance of three modes of partitioning vertices between OpenMP threads demonstrated in [3, 4], here running natively on Xeon Phi. It is clear that MeTiS considerably outperforms and

Figure 9: Strong thread scaling of the offload baseline execution of the flux kernel compared to the native baseline execution (22,677 mesh vertices)

scales well with increasing number of cores compared to others. The upturn of the redundant work timings for more than eight cores was not observed with the small core counts of [3], and this approach is not advocated in the many-core regime.

6.3.1. Cache-Coherency Performance Challenges

Intel Phi uses an extended protocol for the cache-coherency implementation. Intel has modified the shared state (S) in the standard MESI cache protocol to allow modified and unmodified cache-lines in the shared state (S); this approach condenses broadcast storms on the address buses. Addition-

Figure 10: Strong thread scaling of the offload fine-tuned execution of the flux kernel compared to the native fine-tuned execution (22,677 mesh vertices)

ally, Phi's Distributed Tag Directory (DTD)⁸ maintains the global coherency by holding the extended shared state (S), which is Globally Owned, Locally Shared (GOLS) coherency state of each cache-line. Each cache-line miss request passes through the DTD, and the DTD answers based upon the GOLS state, by either fetching the cache-line from the memory or from the core that owns the requested cache-line [35]. Figure 12 shows a flowchart that represents the processes of data transmission between the cores and the hierarchical memory subsystem of Phi.

DTD allows the cores to communicate without having performance deviations necessitated by distance variations between cores. Thus, the distance

⁸Phi's DTD is explained in Appendix B.

Figure 11: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel running on Xeon Phi (Native Mode) parallelized using three different strategies for assignment of vertices to threads (22,677 mesh vertices)

between Phi's cores are extraneous to the performance. Hence, it creates a homogeneous distribution of the shared cache-lines among the cores to avoid performance degradation. This approach, on the other hand, does not take advantage of data locality in the network [45]. Nevertheless, it may be related to the address for the cache miss. The core that has the cache miss event sends the access request to the related DTD through the ring. The implicit communication in transferring cache-lines between cores inhibits understanding and further improving the performance.

6.4. Offload and Native Mode Performance Results With the Fine Mesh

Figure 13 shows a scalability study of running the shared-memory optimized version of PETSc-FUN3D with a more highly resolved mesh (357,900 vertices) on the Phi using native and offload mode. We observe that our

Figure 12: Cache-to-cache transmission scheme of Intel Xeon Phi

optimized version of PETSc-FUN3D still scales well with increasing number of threads even with larger problem sizes as long as they fit into Phi memory.

6.5. Comparison of Optimized Phi Performance to CPU Performance

The best Xeon Phi MIC performance, when we execute 240 thread contexts natively on Xeon Phi, surpasses a single core of the Sandy Bridge by about 8.5x. However, the Sandy Bridge strong scales well on our test case out to 16 cores at one thread per core with a speedup of 12.2x out of 16, and thus beats the MIC by a ratio of 1.4x; see Figure 14. In addition, we compare the baseline of the flux kernel running on the Sandy Bridge with our fine-tuned version, and the results show that the fine-tuning optimizations that are performed on Phi improve the performance of Sandy Bridge; see Figure 15. This leaves it essentially on par with the MIC when both devices are used with the maximum possible number of threads. Since the Sandy Bridge E5-2670 draws a nominal 115W, only 38% of that of the Xeon Phi 7120P, which draws a nominal 300W, and since the Sandy Bridge also retails

Figure 13: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel running on Xeon Phi (Native Mode) compared to the offload mode (357,900 mesh vertices)

for about 38% of the price (a nominal \$1,556 versus \$4,129), it is preferred for our unstructured CFD problem today. However, multi-core performance saturates the hardware prior to the exhaustion of concurrency in the application. All architectural trends indicate the importance of porting to many-core accelerators. This study must be regarded as a preliminary benchmark on an early version of the many integrated core hardware/software environment.

To investigate the strong scalability of our fine-tuned flux kernel on a cutting-edge multi-core CPU architecture, we run the optimized version herein on a Haswell-based Intel Xeon CPU with two different mesh sizes, and we compare the results with Sandy Bridge performance; see Figures 16 and 17. These results show that the aggressive optimizations that have been employed to improve the compute-bound kernel on the Intel Phi also pay off on multicore chips.

Figure 14: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel running a Sandy Bridge-based Intel Xeon CPU compared to Xeon Phi native mode (22,677 mesh vertices)

6.6. Large-scale Strong Scalability Study

As mentioned earlier, the hybrid programming paradigm (MPI+OpenMP) is a means for the PDE-based computations to exploit thousands of compute nodes each of which is connected to a many- or multi-core hardware, as explored previously [1, 2, 3, 4]. Here we reevaluate this claim on our fine-tuned version of the flux kernel of PETSc-FUN3D by running the code on the KAUST's Cray XC40 system, Shaheen II, with two different strong scalability studies, with different experimental settings, as follows.

In the first study, we launch one MPI rank for each compute node and 32 OpenMP threads (16 threads for each Haswell socket) using scatter mode. We disable Intel hyper-threading mode, so that we have one hardware thread is launched by each core. Figure 18 shows the results of the strong scalability

Figure 15: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel running a Sandy Bridge-based Intel Xeon CPU compared to the scatter mode of the baseline model running on the same architecture (22,677 mesh vertices)

of this study.

In the Second study, we launch two MPI ranks for each compute node (one MPI rank for each Haswell socket), and 16 OpenMP threads (8 threads for each Haswell socket) using scatter mode. We also disable hyper-threading mode, so that each core launches one OpenMP thread. Figure 19 shows the results of the strong scalability of this study.

In addition, Table 1 compares time-to-solution of both experimental settings, 1 MPI rank and 32 OpenMP threads per node, as well as 2 MPI ranks and 16 OpenMP threads per node, with respect to the number of hardware cores.

Each large-scale case is run for 3 rounds, and in every round, we execute

Figure 16: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel running on a Sandy Bridge-based Intel Xeon CPU using scatter mode compared to the scatter mode running on a Haswell-based Intel Xeon CPU (22,677 mesh vertices)

each case 15 times. So, in total, we do 45 runs for each test case. The large-scale timings reported here in this paper, represent the average runtime for each experiment among the 45 cases. The time-to-solution is representative and reproducible on average, over possible variations from hardware and job location and scheduling within the system.

The large-scale strong scalability results of Shaheen II show that the flux kernel still scales well in the distributed-memory systems, with potential of running thousands of hardware cores simultaneously, until the subdomain on each processor gets too small a volume-to-surface area ratio with the increasing number of the launched nodes. For example, in the largest case, where we launch 98,304 cores of 3,072 compute nodes, each subdomain consists of at most 900 mesh vertices distributed among each node's memory,

Figure 17: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel running on a Sandy Bridge-based Intel Xeon CPU using scatter mode compared to the scatter mode running on a Haswell-based Intel Xeon CPU (357,900 mesh vertices)

and a subdomain is further divided among the cores by MeTiS, which for example, in the case of 32 threads, each core owns at most 28 mesh vertices locally.

The results of both configurations show that having two MPI ranks for each socket improves the performance, which avoids the local communication penalties imposed by the Non-Uniform Memory Access (NUMA) architecture⁹. After 32,768 compute cores the scaling stagnates. Modeling this

⁹NUMA is a configuration of the memory subsystem in multiprocessing computer architecture. In this configuration, the CPU access time of the main memory is highly dependent upon the physical location of the memory chip corresponding to the CPU. As such, the access time to the memory chip owned by a CPU socket is faster than accessing

Figure 18: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel running on Shaheen II supercomputer using scatter mode with 1 MPI rank and 32 OpenMP threads per compute node (2,761,774 mesh vertices)

stagnation from deteriorating computation-to-communication ratio quantitatively is outside of the scope of this work, which focuses on the strong thread scaling within a single node.

7. Concluding Remarks and Future Work

We have demonstrated performance scalings and saturations in porting a compute-intensive computational kernel of an Euler flow code, a typical unstructured PDE application, onto many integrated core hardware. It is shown to be important from execution time and memory perspectives to

memory chips owned by different sockets [46].

Figure 19: Strong thread scaling of the fine-tuned PETSc-FUN3D flux kernel running on Shaheen II supercomputer using scatter mode with 2 MPI ranks and 16 OpenMP threads per compute node (2,761,774 mesh vertices)

explore several options for data placement, thread allocation to cores, and vertex allocation to threads. The generic optimization techniques employed demonstrate significant performance improvement compared to the baseline of the original code. It is important to avoid the data traffic of the offload mode as much as possible and to align memory segments with padding. Future many-core architectures should offer an autonomous native mode, whenever it is natural to exploit the many-core design, perhaps in an overall heterogeneous setting. In the context of hyper-threading, after more than one thread per core, performance seems rather insensitive to the type of allocation of threads to cores for the same number of threads overall, but very sensitive to the allocation of vertices to threads. These conclusions are believed to be representative of unstructured-mesh PDE-based codes generally, and this

	1 MPI $\times$ 32 OpenMP	2 MPI $\times$ 16 OpenMP
# of Cores	Runtime (Seconds)	
512	8.56E+00	7.07E+00
1,024	5.51E+00	3.95E+00
2,048	2.98E+00	2.23E+00
4,096	1.51E+00	1.23E+00
8,192	9.62E-01	5.95E-01
16,384	6.29E-01	3.99E-01
32,768	4.68E-01	3.40E-01
65,536	3.48E-01	2.70E-01
98,304	3.39E-01	2.52E-01

Table 1: Time-to-solution comparisons between single MPI rank per node approach versus two MPI ranks per node approach running on Shaheen II supercomputer (2,761,774 mesh vertices)

work is a stepping stone en route to a port to the Phi in its current and future generations of full PETSc-FUN3D capability, including the solver, and including richer fluid mechanical models with compressibility and viscosity, and richer workloads, such as adjoint-based optimization.

A multi-core processor with powerful CPUs remains today a formidable match for a specialized many-core accelerator with weaker cores in problems that fail to exploit the natural advantages of the latter in the context of predictable uniformity of addressing. Nevertheless, lessons learned and techniques demonstrated at each architectural generation are transferable to future generations, where there will be an increasing premium on fine-tuning the algorithmic balance to the hardware balance, whether accomplished with some degree of automation or through a significant degree of human expertise.

8. Acknowledgments

The authors are very appreciative of collaborations with Intel Research Laboratories, the Extreme Computing Research Center at KAUST, and Professor Rio Yokota of the Tokyo Institute of Technology. Support in the form of computing resources was provided by the KAUST Supercomputing Laboratory, and KAUST Information Technology Research Division.

Appendix A. Euler Flow Governing PDE Equations

The artificial compressibility version of the incompressible Euler formulation employed in FUN3D is described in [11]. These equations are derived from a control volume formulation that relates the rate of change of a vector of Voronoi cell-averaged state variables $\mathbf{q}$ to the flux through the bounding surface. In integral form,

$$V \frac{\partial \mathbf{q}}{\partial t} + \int_{\delta\Omega} \mathbf{f} \cdot \hat{\mathbf{n}} dl = 0, \quad (\text{A.1})$$

where $\hat{\mathbf{n}}$ is the outward-pointing unit normal to cell Ω of volume V . The vector $\mathbf{q}$ and the inviscid flux normal to the control volume $\mathbf{f}$ are given by

$$\mathbf{q} = \begin{bmatrix} p \\ u \\ v \\ w \end{bmatrix} \quad (\text{A.2})$$

and

$$\mathbf{f} \cdot \hat{\mathbf{n}} = \begin{bmatrix} \beta\Theta \\ u\Theta + n_x p \\ v\Theta + n_y p \\ w\Theta + n_z p \end{bmatrix} \quad (\text{A.3})$$

where β is the artificial compressibility parameter, u , v , and w are the Cartesian velocity components in the x , y , and z directions, respectively, p is the pressure, and Θ is the velocity normal to the surface of the control volume, namely, $\Theta \equiv n_x u + n_y v + n_z w$.

Discussion of the wave-upwinded discretization is well covered in [11]. The key takeaway for performance implications is that there are four scalar components and four scalar conservation laws centered at each vertex that densely inter-couple the state vector unknowns.

Appendix B. Intel Many Integrated Core (MIC) Architecture

Intel Xeon Phi “Knights Corner” is a Peripheral Component Interconnect express (PCIe) Many Integrated Core (MIC) die for logically convoluted, high-throughput, highly parallel high-performance scientific applications. Phi is designed for those applications that can exhaustively exploit

vectorization through the SIMD vector instructions, or bounded by memory bandwidth [47]. The current cutting-edge MIC design can accommodate at most 61 compute cores, running at 1.238 GHz clock speed. The design of the cores is based on a modified version of Intel® Pentium 32-bit P54c architecture that has been extended to fully support 64-bit IEEE arithmetic. The SIMD instruction sets in Phi are implemented by the 512-bit VPU, which provides a large fine-grained data parallelism. The VPU has 16 General-Purpose Registers (GPRs), to store data as well as addresses, and it has 32 Floating-Point Registers (FPRs), to store the floating point numbers. The VPU of Phi introduces an additional 218 new SIMD instructions compared to a conventional Xeon CPU. Furthermore, the VPU implements an Extended Math Unit (EMU) to supply a high-throughput hardware implementation of the single precision transcendental functions as elementary functions, which other functions can be derived from [48, 25]. The transcendental functions that are provided by EMU are:

- Reciprocal function: RECIP ($1/x$)
- Reciprocal square root function: RSQRT ($1/\sqrt{x}$)
- Base two logarithm function: LOG2 ($\log_2(x)$)
- Base two exponential function: EXP2 (2^x)

The first three functions (RECIP, RSQRT, and LOG2) take 1 execution cycle, whereas the EXP2 function takes 2 cycles [49].

Each Phi core includes a short in-order, dual issue pipeline, which ensures that the overhead for the branch misprediction is negligible [50]. In addition, every core implements two launch pipes (execution units), U and V pipes; most of the instructions of Phi's VPU is executed in the U pipe. In addition, in each clock cycle the core can execute two instructions concurrently, distributed between the U pipe and the V pipe. Instructions are scheduled in a sequence between the threads of a single core. However, there is no consecutive execution of instructions from the same threads; there must be at least one cycle delay before decoding the next instruction from the same thread (e.g., if the core has launched only one thread out of four, the thread's instructions will be issued at every other cycle). This minimizes the core's pipeline latency and it also preempts the decode stage of the core's pipeline from pruning the coprocessor's clock frequency [29]. Hence, unlike other

Xeon architectures, Phi's hyper-threading should fully exploit the hardware resources, with at least two thread contexts per core [48].

There are five execution pipelines through which all VPU instructions en route to completion. These pipelines are: 1) Double precision pipeline that executes the 64-bit IEEE arithmetics, converts from 64-bit to 32-bit, and executes the double precision comparison instructions. 2) Single precision pipeline, which executes most of the 64-bit integer loads instructions. 3) Mask pipeline for mask instructions. 4) Store pipeline for store instructions. 5) Scatter and gather pipeline, which reads sparse data from the memory into the vector registers, and writes the sparse data back into the memory from the vector registers.

The two cache levels of MIC are each 8-way set associative, and each has 8 reading channels and 8 memory banks¹⁰. The configuration of the L1 caches in Phi is similar to the cutting-edge Intel Xeon CPUs, which is two L1 caches for data and instructions, each of which is a 32 KB size. However, Intel extends the second level caches to have a capacity of 512 KB local to each core, which also can be viewed as a shared last-level cache of 32 MB capacity between all of the cores, via the high-bandwidth bidirectional network that connects all of the L2 caches together with the memory controllers as well as the global tag directories. The Translation Lookaside Buffer (TLB) page table of each cache level is a 4-way set associative that supports 4 KB, 64 KB, and 2 MB page sizes for the L2 caches, 4 KB and 2 MB page sizes for the L1 data caches, as well as 4 KB for the L1 instruction caches. The bidirectional high-performance interconnect link, which connects the Phi's L2 caches, the global tag directories and the DRAM, consists of three independent rings in each direction, as follows:

- Data block ring (64-bytes wide)
 - The data ring block is designed to provide the high-bandwidth data transmission requirement by the large number of cores
- Address ring

¹⁰A memory bank is a logical storage unit that is identified by the Direct Memory Access. In principle, a bank contains number of rows and columns that specifies chunks of storage with size equal to the numbers of column and row bits; memory banks are distributed across several chips. Similarly, caches are divided into several banks, which are addressed successively in the total collection of memory banks.

- Sends memory addresses, and read/write commands
- Acknowledgment ring
 - Sends flow control and coherence messages

The Xeon Phi chip has 8 Direct Memory Access (DMA) controllers, which support 8 32-bit dual PCIe 2nd Generation memory channels (amounting up to 16 memory channels in total, 16 bits coming from one chip and 16 bits coming from the other chip) of a 16 GB GDDR5 DRAM. Each channel has 4 banks and 4 sub-banks, with a total of 16 banks for each channel (16 channels with 16 banks per channel offers 256 GDDR5 DRAM banks in total). The two GDDR5 chips on each channel act in harmony, so they still provide 16 banks for each channel, but the actual size is doubled ($16 \times 2 = 32$ banks). The sustainable memory bandwidth of each core amounts to 8.4 GB/s (512.4 GB/s in total for all 61 cores) neglecting the throughput capacity of the physical memory [51, 52]. Nonetheless, the bandwidth between the cores and the memory controllers is bounded by the bidirectional ring network that links the cores' caches with the memory controllers, which theoretically transfers in excess of 220 GB/s. Hence, the theoretically aggregated memory bandwidth of Xeon Phi amounts to 352 GB/s¹¹ across all memory controllers directly connected to the MIC die.

The MIC architecture implements a data coherency between the hardware cores that are connected to a bidirectional network by a 64 physical global Distributed Tag Directories (DTDs) hooked up to the bidirectional network. In addition, based on the cache-line address, a hash function connects the physical memory to the DTD to be used for cache-line assignment. The overall address space of Phi is divided evenly among the caches of the cores to achieve an equidistribution of the memory addresses. The DTD helps to minimize memory conflicts in the hot spots of a program. Further, because the DTD protocol is based upon the memory addresses, considerable differences in memory access latencies can be observed, based on the DTD owning the cache-line, rather than the distance between the cores. In the traditional Modified, Exclusive, Shared, Invalid (MESI) [46] cache coherency protocol, each cache-line has to be evicted into the main memory before other threads can read or write that line, which makes the intra-node

¹¹ 5.5 GT/s transfer speed $\times$ 16 channels $\times$ 4 Bytes/transfer.

communication between threads complex and expensive. Therefore, Intel introduced an extended version of the MESI protocol that is implemented on MIC to reduce the cost penalty of the intra-node communication. The extended-MESI protocol offers cache-to-cache (some literature refers to it as “core-to-core”) communication, which explicitly reduces the frequency of eviction into the main memory. This is done despite the fact that the overall latency of reading and writing a cache-line is strongly dependent upon the state of the requested cache-line [45]. Furthermore, a recent empirical study on Xeon Phi L2 cache-line miss latencies has presented that a cache-line miss takes on average 250 CPU clock cycles from L2 remote cache to the requester L2 cache, whereas a 302 CPU clock cycles on average is taken from the DRAM to the requester L2 cache [48, 53].

The 61st core of the Xeon Phi runs a micro version of the Linux Operating System (OS), which operates independently as a compute node linked to the main CPU via a PCIe bus, allowing users to run on the Phi natively. Hence, Intel offers a low-level programming model that reduces the overhead of launching offloaded code, while offering extreme bandwidths for data transfer between the device and the host. This approach utilizes both Intel Coprocessor Offload Infrastructure (COI) as well as Symmetric Communication Interface (SCIF) [54].

Appendix B.0.1. Memory Performance

The performance of Intel Xeon Phi has been well investigated in literature. In [52], Saule *et al.* demonstrate the read and write bandwidth by designing a simple sophisticated benchmark on MIC. In the read case, the achieved bandwidth was almost 171GB/s when 61 cores and 4 threads per core are used. This result is achieved with 32-bit integers but uses vectorization capabilities of Intel Phi to load and sum 16 of them at a time (process 512-bits: a full cache-line at once using SIMD instructions). When 61 cores and 2 threads per core are used, the read-bandwidth reached 183 GB/s, aided by enabling hardware prefetching instructions which allow MIC to request data from the main memory before they are actually needed, effectively hiding the memory latency as well as improving the application efficiency. In the case of writes, the peak bandwidth is achieved by allowing simple store operation, disabling the Read For Ownership (RFO) protocol, which forces the coprocessor to bring a cache-line into the cache before being able to write it, and committing writes memory in an arbitrary order using Non Globally Ordered write instructions (NRNGO). This yields almost 160 GB/s.

The STREAM benchmark [55] is studied on the Phi in [56], with the best performance achieved for 60 threads with balanced thread affinity, with an observed bandwidth of more than 156 GB/s. When the number of threads is increased to 240, the bandwidth drops to about 127 GB/s. The bandwidths of the read and write operations are also calculated alone – about 122.1 GB/s with 244 threads for the reads, and for the writes, 62.9 GB/s.

We investigate the memory bandwidth of our Phi chip using STREAM version 5.10 benchmark with `-O3` compiler flag, and 64,000,000 array elements double precision value (8-bytes per array element). Table B.2 shows the STREAM benchmark results of launching 60 threads (one thread per hardware core), as well as 240 threads (enabling hyper-threading, with 4 threads per core) with scatter thread affinity. Overall, Xeon Phi achieves on STREAM benchmark only about 49% of the peak memory bandwidth with 60 physical threads (2.9GB/s per core and 173GB/s in total with 60 cores).

	60 OpenMP Threads	240 OpenMP Threads
STREAM Kernels	Memory Bandwidth (GB/s)	
Copy	168.304	154.029
Scale	163.555	152.018
Add	173.758	141.632
Triad	173.249	143.312

Table B.2: STREAM benchmark running on Intel Xeon Phi

The best performance is achieved with 60 threads rather than 240 threads, confirming the findings of other researchers. This is due to the GDDR5 DRAM structure of the Phi. In the case of 60 OpenMP threads, STREAM kernels issue around 120 memory read requests plus 60 memory write requests, which in total equals to 180 memory access requests, which is less than the total number of banks of all 16 Xeon Phi's memory channels, namely 256. Thus, with 60 OpenMP threads, Xeon Phi maintains a regular memory access pattern that maps all access requests to different banks without conflicts. On the other hand, in the case of 240 OpenMP threads, STREAM kernels issue around 480 memory read requests plus 240 memory write requests. These 720 memory access requests greatly exceed the total number of memory banks. Irregular accesses result that cause mapping conflicts that degrade the performance of the memory bandwidth.

References

- [1] W. D. Gropp, D. K. Kaushik, D. E. Keyes, B. F. Smith, High performance parallel implicit CFD, *Parallel Computing* 27 (2000) 337–362.
- [2] W. K. Anderson, W. D. Gropp, D. K. Kaushik, D. E. Keyes, B. F. Smith, Achieving high sustained performance in an unstructured mesh CFD application, in: *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing, SC'99*, ACM, New York, NY, USA, 1999.
- [3] D. K. Kaushik, D. E. Keyes, S. Balay, B. Smith, Hybrid programming model for implicit PDE simulations on multicore architectures, in: *International Workshop on OpenMP (IWOMP)*, Vol. 6665 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 12–21.
- [4] D. Mudigere, S. Sridharan, A. Deshpande, J. Park, A. Heinecke, M. Smelyanskiy, B. Kaul, P. Dubey, D. Kaushik, D. Keyes, Exploring shared-memory optimizations for an unstructured mesh CFD application on modern parallel systems, in: *Parallel and Distributed Processing Symposium (IPDPS)*, 2015 IEEE International, 2015, pp. 723–732.
- [5] E. Lusk, A. Chan, Early experiments with the OpenMP/MPI hybrid programming model, in: *OpenMP in a New Era of Parallelism*, Springer, 2008, pp. 36–47.
- [6] R. Rabenseifner, G. Hager, G. Jost, Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes, in: *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, PDP '09*, IEEE Computer Society, Washington, DC, USA, 2009, pp. 427–436.
- [7] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, S. Kumar, E. Lusk, R. Thakur, J. L. Träff, MPI on a million processors, in: *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 20–30.
- [8] R. Thakur, P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, J. L. Traeff, MPI at Exascale, in: *Proceedings of SciDAC 2010*, Vol. 2, 2010, pp. 14–35.

- [9] T. D. Economou, F. Palacios, J. J. Alonso, G. Bansal, D. Mudigere, A. Deshpande, A. Heinecke, M. Smelyanskiy, Towards high-performance optimizations of the unstructured open-source SU2 suite, in: AIAA Infotech @ Aerospace, AIAA SciTech, American Institute of Aeronautics and Astronautics, 2015.
- [10] W. K. Anderson, D. L. Bonhaus, An implicit upwind algorithm for computing turbulent flows on unstructured grids, *Computers and Fluids* 23 (1994) 1–21.
- [11] W. K. Anderson, R. D. Rausch, D. L. Bonhaus, Implicit/Multigrid algorithms for incompressible turbulent flows on unstructured grids, *Journal of Computational Physics* 128 (1996) 391–408.
- [12] W. K. Anderson, P. Balakumar, K. Bibb, B. Biedron, J. Carlson, M. Carpenter, J. Derlaga, P. Gnoffo, D. Hammond, W. Jones, B. Kleb, B. Lee-Rausch, E. Nielsen, H. Nashikawa, M. Park, C. Rumsey, J. Thomas, V. Vatsa, J. White, FUN3D Web page, <http://fun3d.larc.nasa.gov/>, accessed: April 18, 2015 (2015).
- [13] A. Duffy, D. Hammond, E. Nielsen, Production Level CFD Code Acceleration for Hybrid Many-core Architectures, NASA technical memorandum, 2012.
- [14] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, H. Zhang, PETSc Web page, <http://www.mcs.anl.gov/petsc>, accessed: April 18, 2015 (2015).
- [15] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 3.5, Argonne National Laboratory, accessed: September 19, 2014 (2014).
- [16] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: *Modern Software Tools in Scientific Computing*, Birkhäuser Press, 1997, pp. 163–202.

- [17] S. Bhowmick, D. Kaushik, L. McInnes, B. Norris, P. Raghavan, Parallel adaptive solvers in compressible PETSc-FUN3D simulations, in: Proceedings of the 17th International Conference on Parallel CFD, Elsevier, 2005.
- [18] D. E. Keyes, D. K. Kaushik, B. F. Smith, Prospects for CFD on petaflops systems, in: P. Bjorstad, M. Luskin (Eds.), Parallel Solution of Partial Differential Equations, Vol. 120 of The IMA Volumes in Mathematics and its Applications, Springer, 2000, pp. 247–277.
- [19] D. A. Knoll, D. E. Keyes, Jacobian-free Newton-Krylov methods: A survey of approaches and applications, *Journal of Computational Physics* 193 (2) (2004) 357–397.
- [20] W. Gropp, D. E. Keyes, L. C. McInnes, M. D. Tidriri, Globalized Newton-Krylov-Schwarz algorithms and software for parallel implicit CFD, *International Journal of High Performance Computing Applications* 14 (2000) 102–136.
- [21] T. S. Coffey, C. T. Kelley, D. E. Keyes, Pseudo-transient continuation and differential-algebraic equations, *SIAM Journal on Scientific Computing (SISC)* 25 (2003) 553–569.
- [22] C. Kelley, D. E. Keyes, Convergence analysis of pseudo-transient continuation, *SIAM Journal on Numerical Analysis* 35 (2) (1998) 508–523.
- [23] Intel Xeon Phi coprocessor instruction set architecture reference manual, <https://software.intel.com/sites/default/files/forum/278102/327364001en.pdf>, accessed: December 25, 2014.
- [24] D. M. Tullsen, S. J. Eggers, H. M. Levy, Simultaneous multithreading: Maximizing on-chip parallelism, in: Proceedings 22nd Annual International Symposium on Computer Architecture, 1995, pp. 392–403.
- [25] J. Jeffers, J. Reinders, Intel Xeon Phi Coprocessor High Performance Programming, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2013.
- [26] Intel Xeon Phi coprocessor system software developers guide, <https://software.intel.com/sites/default/files/managed/09/07/xeon-phi-coprocessor-system-software-developers-guide.pdf>, accessed: December 25, 2014.

- [27] G. Lawson, M. Sosonkina, Y. Shen, Energy evaluation for applications with different thread affinities on the Intel Xeon Phi, in: International Symposium on Computer Architecture and High Performance Computing Workshop (SBAC-PADW), IEEE, 2014, pp. 54–59.
- [28] Cornell virtual workshop, accessed: September 16, 2014.
URL <https://www.cac.cornell.edu/vw/mic/default.aspx>
- [29] R. Krishnaiyer, E. Kultursay, P. Chawla, S. Preis, A. Zvezdin, H. Saito, Compiler-based data prefetching and streaming non-temporal store generation for the Intel[®] Xeon Phi[™] coprocessors, in: IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), 2013, pp. 1575–1586.
- [30] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on Scientific Computing* 20 (1) (1998) 359–392.
- [31] Y. Saad, M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM Journal on Scientific and Statistical Computing* 7 (3) (1986) 856–869.
- [32] J. Anderson, *Computational Fluid Dynamics, Computational Fluid Dynamics: The Basics with Applications*, McGraw-Hill Education, 1995.
- [33] I. Z. Regulý, E. Lszl, G. R. Mudalige, M. B. Giles, Vectorizing unstructured mesh computations for many-core architectures., *Concurrency and Computation: Practice and Experience* 28 (2) (2016) 557–577.
- [34] User and reference guide for the Intel[®] Fortran compiler 15.0, accessed: October 22, 2015.
URL https://software.intel.com/en-us/compiler_15.0_ug_f?language=ru
- [35] R. Rahman, *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*, 1st Edition, Apress, Berkeley, CA, USA, 2013.
- [36] A. S. Tanenbaum, *Modern Operating Systems*, 3rd Edition, Prentice Hall Press, Upper Saddle River, NJ, USA, 2007.

- [37] M. Sabahi, A guide to auto-vectorization with Intel® C++ compilers, <https://software.intel.com/en-us/articles/a-guide-to-auto-vectorization-with-intel-c-compilers>, accessed: August 26, 2015 (2012).
- [38] E. Cuthill, J. McKee, Reducing the bandwidth of sparse symmetric matrices, in: Proceedings of the 1969 24th National Conference, ACM'69, ACM, New York, NY, USA, 1969, pp. 157–172.
- [39] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy, S. A. Jarvis, Exploring SIMD for molecular dynamics, using Intel®Xeon®processors and Intel®Xeon Phi™ coprocessors, in: Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, IPDPS '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 1085–1097.
- [40] M. B. Giles, G. R. Mudalige, Z. Sharif, G. Markall, P. H. Kelly, Performance analysis of the OP2 framework on many-core architectures, ACM SIGMETRICS Performance Evaluation Review 38 (4) (2011) 9–15.
- [41] Z. Bozkus, A. Choudhary, G. Fox, T. Haupt, S. Ranka, Fortran 90D/HPF compiler for distributed memory MIMD computers: Design, implementation, and performance results, in: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, SC'93, ACM, 1993, pp. 351–360.
- [42] R. Aubry, G. Houzeaux, M. Vazquez, J. Cela, Some useful strategies for unstructured edge-based solvers on shared memory machines, International journal for numerical methods in engineering 85 (5) (2011) 537–561.
- [43] C. J. Newburn, R. Deodhar, S. Dmitriev, R. Murty, R. Narayanaswamy, J. Wiegert, F. Chinchilla, R. McGuire, Offload compiler runtime for the Intel® Xeon Phi™ coprocessor, in: Supercomputing - 28th International Supercomputing Conference, ISC 2013, Leipzig, Germany, June 16-20, 2013. Proceedings, 2013, pp. 239–254.
- [44] C. J. Newburn, S. Dmitriev, R. Narayanaswamy, J. Wiegert, R. Murty, F. Chinchilla, R. Deodhar, R. McGuire, Offload compiler runtime for the Intel® Xeon Phi™ coprocessor, in: 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013, 2013, pp. 1213–1225.

- [45] S. Ramos, T. Hoefer, Modeling communication in cache-coherent SMP systems: A case-study with Xeon Phi, in: Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing, HPDC'13, ACM, New York, USA, 2013, pp. 97–108.
- [46] J. L. Hennessy, D. A. Patterson, Computer Architecture, Fifth Edition: A Quantitative Approach, 5th Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.
- [47] X. Tian, H. Saito, S. V. Preis, E. N. Garcia, S. S. Kozhukhov, M. K. Masten, A. G. Cherkasov, N. Panchenko, Practical SIMD vectorization techniques for Intel[®] Xeon Phi[™] coprocessors, in: IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), 2013, pp. 1149–1158.
- [48] J. Fang, A. L. Varbanescu, H. J. Sips, L. Zhang, Y. Che, C. Xu, An empirical study of Intel Xeon Phi, CoRR abs/1310.5842 (2013) 1–11.
- [49] S. Li, Achieving superior performance on Black-Scholes valuation computing using Intel[®] Xeon Phi[™] coprocessors, <https://software.intel.com/en-us/articles/case-study-achieving-superior-performance-on-black-scholes-valuation-computing-using>, accessed: July 6, 2015.
- [50] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, Y. Wang, High-Performance Computing on the Intel Xeon Phi[™]: How to Fully Exploit MIC Architectures, Springer International Publishing, 2014.
- [51] C. Calvin, F. Ye, S. Petiton, The exploration of pervasive and fine-grained parallel model applied on Intel Xeon Phi coprocessor, in: Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013, IEEE, Compiegne, France, 2013, pp. 166–173.
- [52] E. Saule, K. Kaya, U. Catalyurek, Performance evaluation of sparse matrix multiplication kernels on Intel Xeon Phi, in: Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 559–570.
- [53] Y. Wang, M. J. Anderson, J. D. Cohen, A. Heinecke, K. Li, N. Satish, N. Sundaram, N. B. Turk-Browne, T. L. Willke, Full correlation matrix

- analysis of fMRI data on Intel® Xeon Phi™ coprocessors, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, ACM, New York, NY, USA, 2015, pp. 23:1–23:12.
- [54] X. Dong, J. Chai, J. Yang, M. Wen, N. Wu, X. Cai, C. Zhang, Z. Chen, Utilizing multiple Xeon Phi coprocessors on one compute node, in: Algorithms and Architectures for Parallel Processing, Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 68–81.
- [55] J. D. McCalpin, STREAM: Sustainable memory bandwidth in high performance computers, Tech. rep., University of Virginia, Charlottesville, Virginia, accessed: October 11, 2014 (1991-2007).
URL <http://www.cs.virginia.edu/stream/>
- [56] T. Cramer, D. Schmidl, M. Klemm, D. an Mey, OpenMP programming on Intel® Xeon Phi™ coprocessors: An early performance comparison, in: Proceedings of the Many-core Applications Research Community (MARC) Symposium, RWTH Aachen University, Aachen, Germany, 2012, pp. 38–44.