Early Application Experiences with the Intel MIC Architecture in a Cray CX1

R. Glenn Brook, Bilel Hadri, Vincent C. Betro, Ryan C. Hulguin, Ryan Braby
National Institute for Computational Sciences
University of Tennessee
Oak Ridge National Laboratory
Oak Ridge, TN USA
Email: glenn-brook@tennessee.edu, bhadri@utk.edu, vbetro@utk.edu, ryan-hulguin@tennessee.edu, rbraby@utk.edu

***Abstract*—This work details the early efforts of the National Institute for Computational Sciences (NICS) to port and optimize scientific and engineering application codes to the Intel Many Integrated Core (Intel MIC) architecture in a Cray CX1. After the configuration of the CX1 is presented, the successful portings of several application codes are described, and scaling results for the codes on the Intel Knights Ferry (Intel KNF) software development platform are presented.**

***Index Terms*—Intel MIC, Cray CX1, applications, porting**

I. Introduction

With the stagnation of computer processor frequency due to the constraints of power and density, processor manufacturers are introducing additional parallelism within their designs to attain increased performance for highly parallel applications. General-purpose graphical processing units (GPGPUs) employed in most modern accelerators rely on a modest number of streaming multiprocessors that perform extremely wide vector calculations, expressed using a specialized (and largely proprietary) programming model, to achieve high theoretical peak performance. The Intel Many Integrated Core (Intel MIC) architecture takes a different approach, instead relying on a massive number of parallel, x86-compatible cores equipped with high performance, wide vector processing units to achieve similar performance using standard, parallel programming models and languages. This design approach offers several unique advantages that are expected to provide a much more direct and cost-effective path for migrating existing scientifc and engineering codes to accelerated architectures than that currently available with GPGPUs.

To explore the potential impact of the Intel MIC architecture and related programming models on the scientific supercomputing community, several applications are ported to and subsequently studied on two 32-core Intel Knights Ferry (Intel KNF) software development platforms deployed in a Cray CX1 at the National Institute for Computational Sciences (NICS). Three of the studied applications – an Euler solver, a Boltzmann-BGK solver, and a Navier-Stokes solver – employ the native execution model, in which the Intel MIC acts as a standalone compute node on which parallel code is executed directly via MPI or interactive login. The final application – a Poisson solver – employs the offload execution model, in which computational kernels or sections of code that are flagged by special pragmas are automatically migrated to the Intel MIC for execution while the primary code runs on the host processor(s). Information regarding the porting, application, and scalability of each application code is presented, following a brief discussion of the configuration of the Cray CX1.

II. Intel MIC configuration on the Cray CX1

The Cray CX1 system has one management node and two compute nodes. Each of the compute nodes has two Intel Xeon 5600 processors and a single 32-core Intel KNF card. Following initial configuration of the host and compute nodes by Cray, NICS staff installed the drivers, compilers, and support software for the Intel KNF cards. These cards are supported by early Alpha software; but, despite the limitations of this early software, the CX1 and its Intel KNF cards are surprisingly stable and usable.

Each Intel KNF card boots a diskless Linux OS image when initialized. This image has a virtual IP interface between it and the host node, and it runs a ssh server that listens on this interface. In the alpha releases of the software, no accounts other than root are available on the Intel KNF card. So, each user you want to enable to ssh into an Intel KNF card needs a copy of the root ssh keys. Applications that have been compiled for the Intel MIC can then be copied over to a Intel KNF card and run in native mode.

Running MPI applications in native mode across multiple Intel KNF cards requires a method of communication between the cards in the different compute nodes. This is accomplished by setting up TCP/IP communications between the cards. Since the alpha software includes a default IP address for the Intel KNF card card and does not support a method for changing this on card initialization, Network Address Translation (NAT) is used to make it appear that the Intel KNF cards are on the same ethernet network as their host nodes. Each Intel KNF card requires the following three NAT rules:

1) -A PREROUTING -d [virtual address for KNF/mask] -j DNAT –to-destination 192.168.1.100
2) -A POSTROUTING -s 192.168.1.100/32 -j SNAT –to-source [virtual address for MIC]
3) -A OUTPUT -d [virtual address for MIC/mask] -j DNAT –to-destination 192.168.1.100

In addition, each Intel KNF card requires a default route to the virtual interface on the host node. Each host node needs IP forwarding enabled to allow for traffic to pass between the virtual network connecting it and the Intel KNF it hosts to the ethernet network connecting the nodes together. When all of this is in place (and scripts are setup to reconfigure it after each boot), the Intel KNF cards and compute nodes can all ping each other, and applications built with MPI using IP communications can communicate between the Intel KNF cards. However, passwordless ssh between all of the hosts needs to be configured to allow for task launching and control.

Copying the ssh keys from the Intel KNF cards to the host nodes and syncing them across the nodes, enables users to login to the OS on the Intel KNF cards. However, going from one Intel KNF to another requires using the dropbearconvert command to make the keys compatible with the dropbear ssh on the Intel KNF cards. Like the networking changes above, this needs to be put in place by scripts after each boot of an Intel KNF card.

Much of this configuration work is necessary to work around limitations of the early alpha software, and it is expected that much of it will be handled differently as newer versions of the software are made available. That said, the quick configuration and work arounds allow early users to accomplish significant work on this CX1 test system.

III. Solving the Euler Equations and the BGK Model Boltzmann Equation

Two computational fluid dynamics (CFD) solvers are developed for use on the Intel MIC architecture. The first solver is based on the Euler equations $(1-5)$, and the second solver is based on the BGK model Boltzmann equation $(6-8)$. These particular solvers are vastly different from each other in terms of their target applications and in their complexity. The Euler equations are typically used to solve inviscid fluid flows, while the BGK model Boltzmann equation is typically used to solve non-continuum rarefied gas flow. The Euler equations only need 5 state variables to be solved at each grid point, whereas the BGK model Boltzmann equation could have hundreds of thousands of state variables that need to be solved at each grid point. Despite their differences, the same numerical algorithm can be used to solve both sets of equations. A simple test problem is simulated using each solver, and a strong scaling study is performed to see how well the solvers scale across the cores of the KNF card.

A. Governing Equations

The Euler equations are shown below

$$\frac{\partial \vec{Q}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial y} + \frac{\partial \vec{H}}{\partial z} = \vec{0} \tag{1}$$

$$\vec{Q} = [\rho, \rho u, \rho v, \rho w, E]^T \tag{2}$$

$$\vec{F} = [\rho u, \rho u^2 + P, \rho uv, \rho uw, (E+P)\,u]^T \tag{3}$$

$$\vec{G} = [\rho v, \rho uv, \rho v^2 + P, \rho vw, (E+P)\,v]^T \tag{4}$$

$$\vec{H} = [\rho w, \rho uw, \rho vw, \rho w^2 + P, (E+P)\,w]^T \tag{5}$$

where ρ is the density $\vec{u} = (u, v, w)$ are the velocities, P is the pressure and E is the total energy per unit volume.

The BGK model Boltzmann equation using a discrete velocity model [1] is shown below

$$\frac{\partial f_k}{\partial t} + \vec{V_k} \cdot \frac{\partial f_k}{\partial \vec{x}} = \nu \cdot (f_k^{eq} - f_k) \tag{6}$$

$$f^{eq} = \frac{n}{(\pi T)^{3/2}} \left(-\frac{\left(\vec{V_k} - \vec{u}\right)^2}{T} \right) \tag{7}$$

$$\nu = \frac{8nT^{1-\omega}}{5\sqrt{\pi}\mathrm{Kn}} \tag{8}$$

where f_k is the probability distribution function at discrete velocity $k, \vec{V_k} = (V_{x_k}, V_{y_k}, V_{z_k})$ are the discrete molecular velocities, $\vec{u} = (u, v, w)$ are the bulk (average) velocities, n is the number density, T is the temperature, ω is the gas viscosity index, and Kn is the Knudsen number, which characterizes the flow. The Knudsen number is defined as the ratio between the mean free path λ_∞ and the characteristic length L. When the Knudsen number is larger than 0.1, the continuum assumption that the Navier-Stokes equations are based on starts to break down, and the Boltzmann equation becomes more appropriate to use.

B. Numerical Algorithm

The Euler solver and the BGK model Boltzmann solver are both developed using a point-iterative Jacobian-free Newton algorithm [2]. Newton's method is used to linearize the nonlinear system of equations, and then the Jacobi method is used to solve the resulting linear system. The Jacobian, J, is calculated implicitly through the use of dual numbers and Taylor series expansions in the dual space. A dual number[3], ϵ, is like an imaginary number i, except that $\epsilon^2 \equiv 0$. The implicit Jacobian calculations are shown below

$$J_k \left(\Delta q^p\right)^{m-1} \approx \frac{1}{h}\mathrm{Dual}\left[F_k\left(\vec{q}^{\,p-1} + \epsilon h \left(\Delta \vec{q}^{\,p}\right)^{m-1}\right)\right] \tag{9}$$

$$F_k \approx \mathrm{Real}\left[F_k\left(\vec{q}^{\,p-1} + \epsilon h \left(\Delta \vec{q}^{\,p}\right)^{m-1}\right)\right] \tag{10}$$

$$J_{kk} \approx \frac{1}{h}\mathrm{Dual}\left[F_k\left(\vec{q}^{\,p-1} + \epsilon h \vec{e}_k\right)\right] \tag{11}$$

where $\vec{F}(\vec{q}) = \vec{0}$ is the nonlinear set of equations being solved with $1 < k < n$, p is the Newton iteration, m is the current Jacobi iteration, $h \in \mathbb{R}$ is an arbitrary perturbation, and e^k is the kth vector in the standard basis for $\mathbb{R}^N$. The iterative process continues until either a convergence tolerance is met or a maximum number of iterations have completed. The convergence tolerance values are $\|\Delta q\|_2 = 1 \times 10^{-15}$ and $\|\Delta\left[\Delta q\right]\|_2 = 1 \times 10^{-8}$ for the Newton and Jacobi methods respectively, and the maximum number of iterations are 15 Newton iterations with 3 Jacobi iterations per Newton iteration.

C. Parallel Implementation

The Euler solver and the BGK model Boltzmann solver are both compiled in native mode for use on a single KNF card directly. OpenMP threads are used to achieve data parallelism across the cores of the KNF card. The main computation loop over all the grid points is made parallel through the use of the `#pragma omp parallel for` directive. Since the Jacobi method is used to solve the linearized set of equations, the majority of the calculations become almost embarrassingly parallel. Additional loops are also made parallel using OpenMP directives.

D. Test Problems

The test problem run using the Euler solver is the Sod shock [4] simulation. In a shock wave, the properties of a fluid change almost instantaneously. The standard Sod shock starts off with a fluid at rest with the following initial conditions:

$$\rho_{\text{left}} = 0.0, u_{\text{left}} = 0.0, P_{\text{left}} = 1.0$$

$$\rho_{\text{right}} = 0.125, u_{\text{right}} = 0.0, P_{\text{right}} = 0.1$$

The Sod shock is a popular test case for verifying a solver's ability to appropriately capture shocks and contact discontinuities in unsteady fluid flows. The solution generated on the KNF is shown below

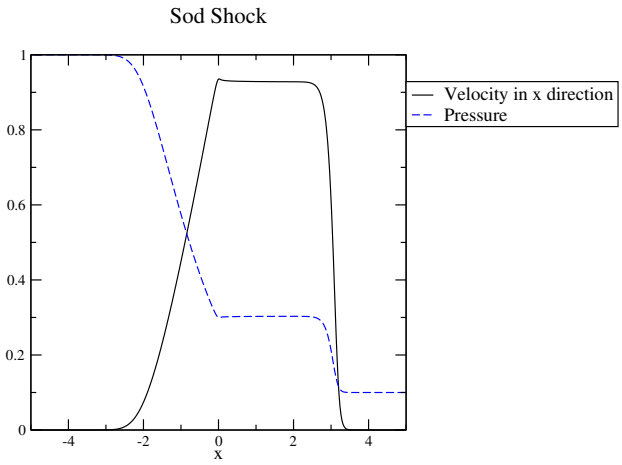


Fig. 1: Numerical simulation of the Sod shock test problem

The test problem run using the BGK model Boltzmann solver is a Couette flow simulation. In a Couette flow, gas is initially at rest between two infinitely long parallel plates. For this simulation [5], the left plate is stationary while the right plate moves at 300 m/s. The gas between the plates is initialized with the following values:

$$\rho_0 = 9.28 \times 10^{-8}\, kg/m^3, u_{x0} = u_{y0} = 0.0\, m/s$$

$$T_0 = 273.0\,\text{K}, \text{Kn} = 0.1199$$

Both plates are held at a constant temperature of 273.0 K. Over time, the gas settles into a steady state solution. The Couette flow problem is a good test case for verifying a solver's ability to handle solid surfaces and moving boundary conditions. The test problem was simulated on a grid with 27 grid points in physical space and 36x36x36 grid points in velocity space. The steady state velocity profile is shown in Figure 2. Note that there is some slip at both walls. This is in contrast to a continuum solution, where the velocities near the left and right walls would be 0.0 m/s and 300 m/s respectively.

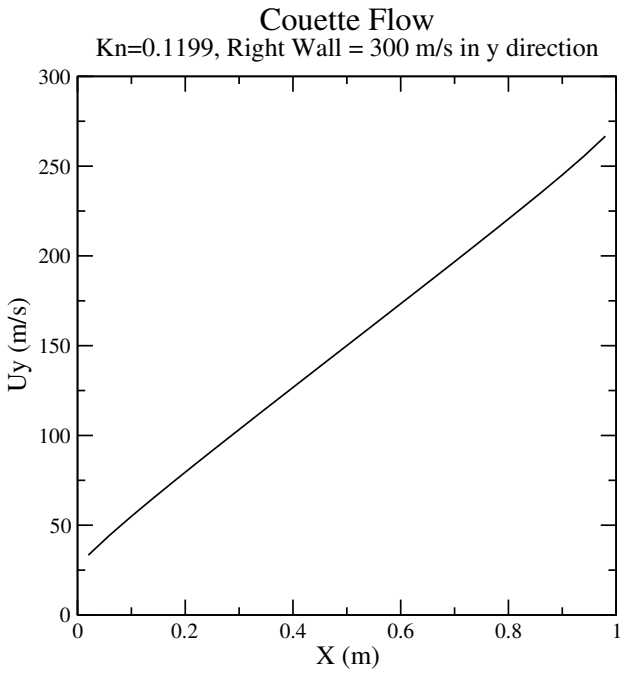


Fig. 2: Numerical simulation of the Couette flow test problem

E. Parallel Performance

To see how well the solvers scale across the cores of the KNF, each test problem was run with a varying number of threads ranging from 1-120. The strong scaling study for the Euler solver is shown in Figure 3, and the strong scaling study for the BGK model Boltzmann solver is shown in Figure 4. The Euler solver scaled very well across the KNF, obtaining a speedup of 30.2 when 108 threads are used. The BGK model Boltzmann solver did not scale well initially, because there was not enough parallelism exposed. Rob Van der Wjingaart, a senior software engineer from Intel, performed some optimizations on the BGK model Boltzmann solver. The strong scaling study is run again with optimizations, and the scalability becomes similar to that achieved with the Euler solver. The optimizations performed include fusing loops to expose more parallelism, vectorizing loops through alignment and compiler directives, and reducing the number of parallel sections. The optimized BGK model Boltzmann solver obtains a speedup of 30.2 when 76 threads are used.

IV. Solving the Navier-Stokes Equations

Two MPI parallel Navier-Stokes (NS) simulations have been ported and benchmarked on the Cray CX1 using native mode, where the executable is run entirely on the Intel KNF. This

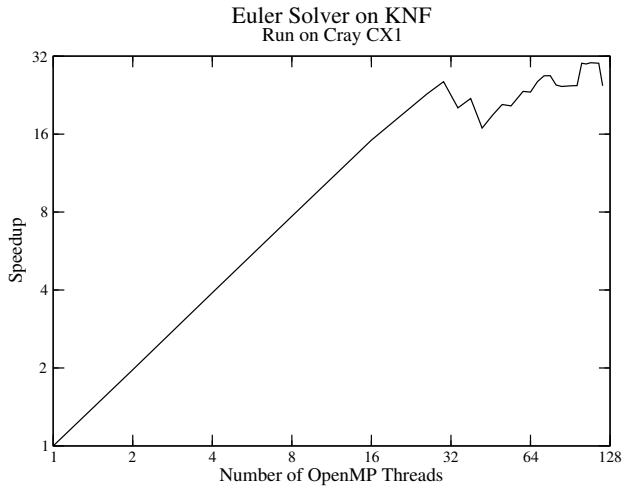


Fig. 3: Strong scaling study for Sod shock test problem

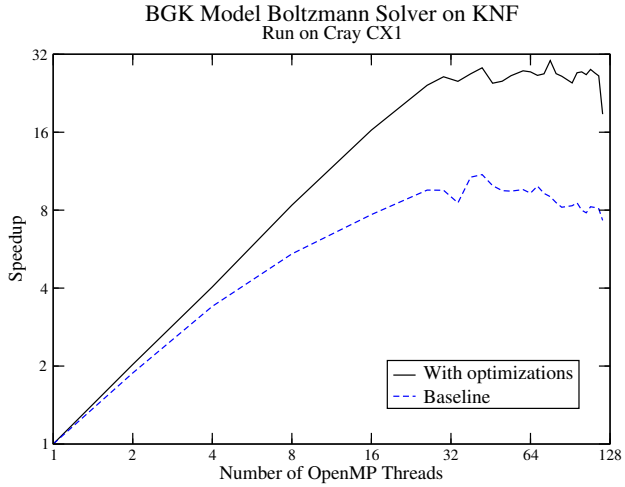


Fig. 4: Strong scaling study for Couette flow test problem

was performed easily by compiling the code with the mpich library with the *-mmic* flag.

The simulations, called Navier Stokes (NS) 2D and 3D, use a domain decomposition technique with a one-dimentional processor topology. The velocity-pressure formulation for the NS equations is employed to obtain the pressure and flow indicators for wall shear stress:

$$\partial_t V + (V \cdot \nabla)V + \nabla p - \nu \nabla \cdot (\nabla V) = f, \text{ in } \Omega$$

$$div(V) \ = \ 0, \text{ in } \Omega$$

where $V = V(X,t)$ is the velocity field at the point $X \in \Omega$, p is the normalized pressure field, f is the source term and ν is the coefficient of kinematic viscosity. Ω denotes the relevant geometrical domain. The second equation corresponds to the incompressibility of the fluid.

The discretization of the NS equations is done with finite differences on a Cartesian grid following the standard staggered grid method. Regarding the resolution of the equation,

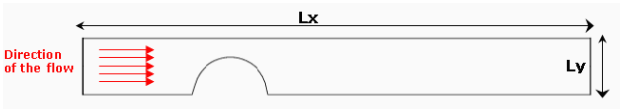


Fig. 5: Benchmark problem

a standard projection method [6] with two steps is employed. First, the prediction of the velocity is performed by solving the momentum equations explicitly; then, the projection of the predicted velocity to the space of divergence free functions is calculated.

A. Hemodynamic with Navier Stokes 2D

1) Description of the problem: The Navier Stokes 2D code simulates a Poiseuille flow, a steady problem in a straight pipe obstructed by an obstacle (the upper half of a disk as described in Figure 5). The velocity profile in the inlet of the pipe corresponds to a Poiseuille flow with a velocity of maximum value one. Inlet and outlet boundary conditions on the flow speed are imposed along the vertical walls at $x = 0$ and $x = L_x$. The flow field at the inlet satisfies the Dirichlet boundary condition

$$u_1(0, y, t) = g(y, t), \ y \in (0, L_y),$$

where g is set to zero inside the wall, i.e for y such that $(0, y) \in \Omega_w$.

For simplicity a homogeneous Neumann boundary condition is imposed on the flow field, i.e $\frac{\partial u_1}{\partial x} = \frac{\partial u_2}{\partial x} = 0$ at the outlet $x = L_x$ and a constant profile of the pressure along the outlet wall $x = L_x$.

2) Parallel paradigm: Domain decomposition with Aitken Schwartz: As described in [7], domain decomposition (DD) techniques can be used as a process of distributing a computational domain among a set of interconnected processors for the coupling of different physical models applied in different regions of a computational domain. In all cases, DD methods:

- rely on a partitioning of the computational domain into subdomains,
- solve in parallel the local problems using a direct or iterative solver, and
- call for an iterative procedure to combine the local solutions to obtain the solution of the global (original) problem.

DD techniques have received much interest in recent years. Indeed, they are suitable on multiprocessor computing systems since they solve restrictions of the problem on different subdomains independently and then integrate the partial solutions. It is known that Additive Schwarz has a very slow convergence rate. This can be a major drawback for a parallel application since the algorithm requires communication at each iteration until convergence is reached.

To accelerate the convergence, the Aitken acceleration has been introduced by Garbey and Tromeur Dervout [8, 9] . Aitken Schwarz(AS) is a DD method using the framework of Additive Schwarz and based on an approximate reconstruction

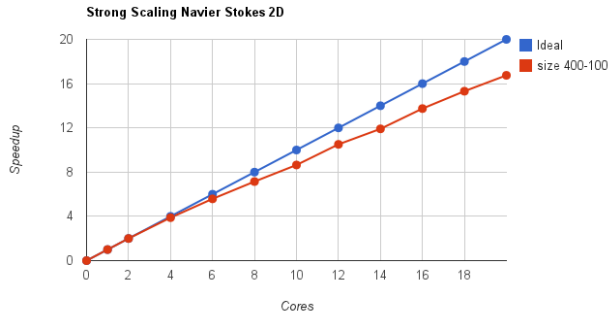


Fig. 6: Strong Scaling

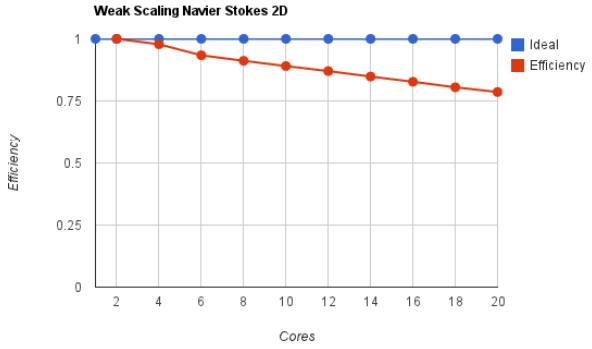


Fig. 7: Weak Scaling

of the dominant eigenvectors of the trace transfer operator. This acceleration technique converges for the Laplacian operator in only two iterations. It is friendly to cache use, and it scales with the memory in parallel environments, providing a fast means to solve the pressure equation. Further details about the approach and its implementations are available in prior works [10, 11].

3) Scaling Results: Figure 6 presents the strong scaling of the parallel NS 2D for a problem size fixed to 400 grid points along the x axis and 100 along the y axis. The speedup is quasi linear up to 6 cores and it reaches a speedup close to 17 when using 20 cores.

Regarding the weak scaling, Figure 7 shows that the code scales well. Indeed, the code employs nearest-neighbour communication patterns where the communication overhead is relatively constant regardless of the number of cores used.

B. Navier Stokes 3D

1) Numerical and parallel approach: The NS solver uses an immersed boundary technique that relies on the proper combination of three techniques. The L_2 penalty approach pioneered by Caltagirone [12, 13] is used to deal with complex geometry for solving the NS equations. Furthermore, it combines nicely with a level set method based on the Mumford-Shah energy model [14] to acquire the geometry of a large vessel from medical images. Finally, we use an AS domain

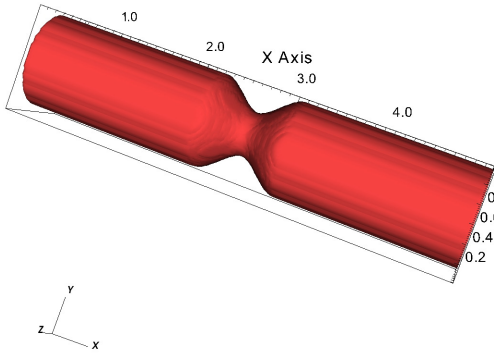


Fig. 8: Geometry

decomposition that has high numerical efficiency and scales well.

The flow solver is an immersed-boundary-like method [15]. The wall boundary condition is immersed in the Cartesian mesh thanks to a penalty term added to the momentum equation. Due to this, there is no need for tightly packed prismatic viscous layers to impose the no slip boundary condition on the wall.

The Aitken-Schwarz Domain decomposition technique is used to solve the pressure equation; however, the main difference between the two and three dimensional simulation is that the Fourier decomposition and the subdomain solver solve only a tridiagonal problem. Indeed, the pressure unknown is developed into the sine expansion in the physical space plane y, z such that

$$p(x,y,z) = \sum_{j=1}^{My}\sum_{k=1}^{Mz} \hat{p}^{j,k}(x).sin(j\pi y).sin(k\pi z)$$

with My and Mz correspondig to the number of modes in the direction of y and z, respectively. Placing this decomposition into the pressure equation results in one dimensional $My \times Mz$ problems based on the Helmotz operator, i.e., a tridiagonal matrix which can be solved with a basic LU decomposition[10]. Besides these changes, the general algorithm is the same as described earlier.

Simulation results visualized with VisIt [16] are presented in Figures 8 to 11.

2) Scaling Results: Unlike the 2D version, the acceleration of Aitken-Schwarz step is done by only one processor, and it is the only step that is not parallelized because of the Aitken acceleration. Amdahl's law[17] states that if P is the proportion of a program that can be made parallel (i.e., benefit from parallelization) and $(1-P)$ is the proportion that cannot be parallelized (remains serial), then the maximum speedup that can be achieved by using N processors is:

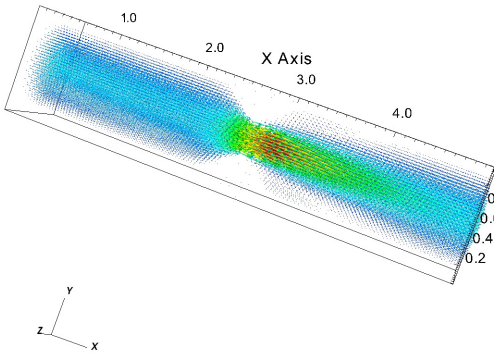


Fig. 9: Velocity magnitude

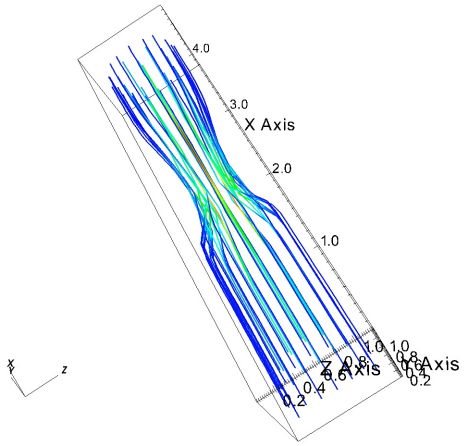


Fig. 10: Particles flow

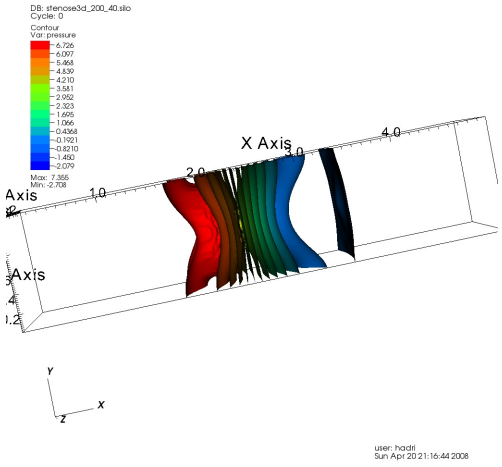


Fig. 11: Pressure field

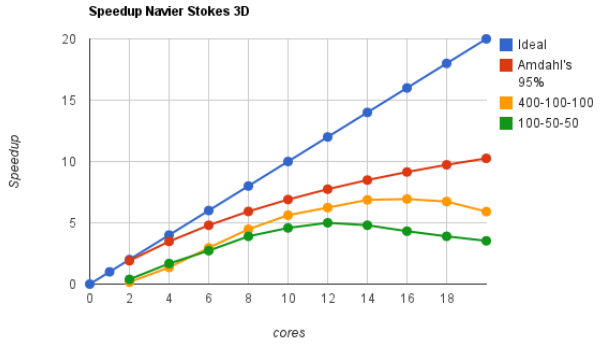


Fig. 12: Strong Scaling

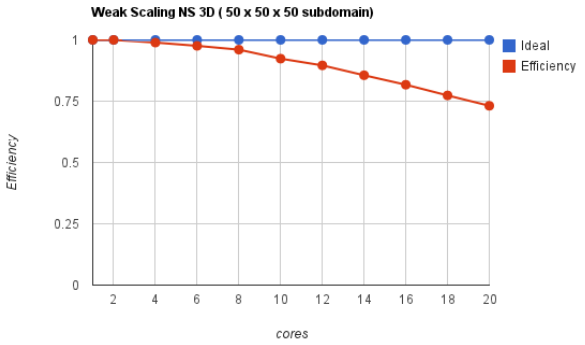


Fig. 13: Weak Scaling

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}.$$

Figure12 shows that when the problem size is large (400 grid points along the x axis, 100 along the y axis z axis), the curve of speedup is close to the Amdahl's law with $P = 0.95$. When the number of cores and subdomains is increased, the size of each subdomain is small, and the performance decreases due to the increased amount of communication. With respect to the weak scaling of the code presented in the Figure13, a reasonable efficiency is observed, with a performance of close to 75% on 20 cores.

V. Solving the Poisson Problem

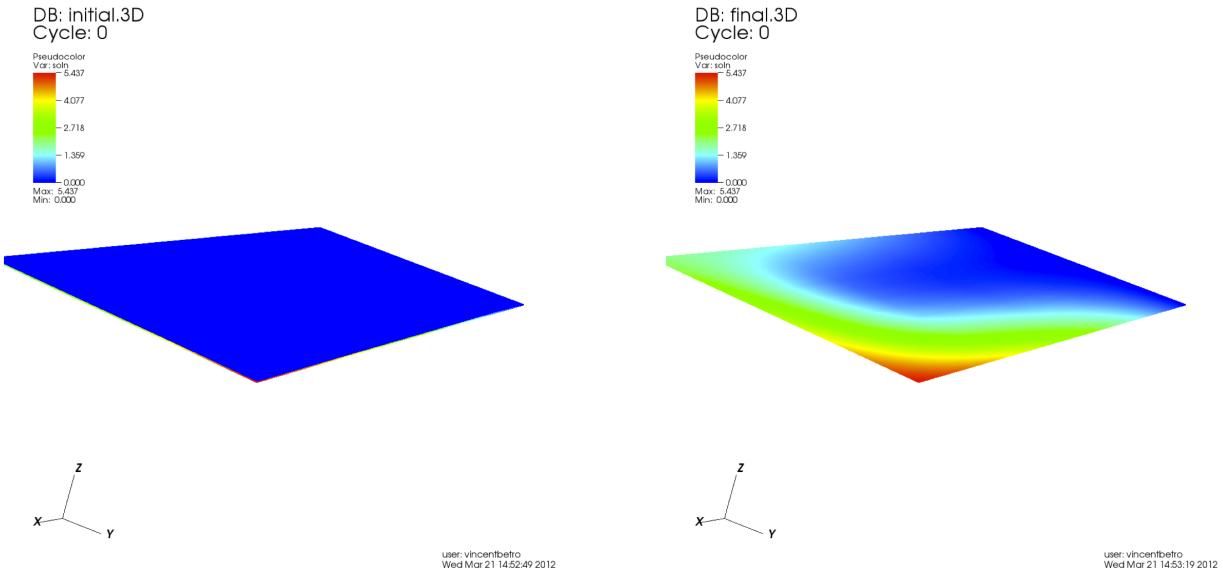
The two dimensional Poisson equation is a partial differential equation as follows:

$$\nabla^2 T = S\,(x, y) \tag{12}$$

where S is an arbitrary source term. A practical application of Poisson's equation is to model thermal conduction on a plate with an added heat source. In this case, the variable T corresponds to the temperature of a point on the rectangular plate subject to Dirichlet (fixed temperature) boundary conditions.

The specific problem being solved in this paper is on the domain

$$0 \leq x \leq 2 \tag{13}$$



(a) Initial Temperatures on Flat Plate

(b) Final Temperatures on Flat Plate

Fig. 14: Simulation results on a 1200 x 1200 flat plate

$$0 \leq y \leq 1 \tag{14}$$

subject to the boundary conditions

$$T(0, y) = 0 \tag{15}$$

$$T(2, y) = 2e^y \tag{16}$$

$$T(x, 0) = x \tag{17}$$

$$T(x, 1) = ex \tag{18}$$

and the source term

$$S(x, y) = xe^y. \tag{19}$$

Three cases distinguised by the number of points (600, 1200, or 2400) in both the x and y directions are considered, and the exact solution to the PDE is $T = xe^y$. A visual of the temperature distribution on the plate both initially and at equilibrium is presented in Figure 14, and there is no appreciable difference in solution values when moving from a 1200x1200 discretization to a 2400x2400 discretization.

A. Mapping Physical Topology to Discretization

Denoting the x direction to be as i and the y direction as j results in a direct mapping of the physical domain to the row and column structure inherent in matrices. Thus, the temperature positions on the flat plate are visualized as being the entries in a matrix overlaying the flat plate. Additionally, the Dirichlet boundary conditions result in the first and last rows and columns of the resulting matrix being static.

B. Gauss-Seidel and Block Jacobi with SSOR Algorithm

Assuming a forward iteration through the mesh, the Gauss-Seidel and symmetric successive over-relaxation (SSOR) scheme depends on having the latest values of T available for $T_{i-1,j}$ and $T_{i,j-1}$, as seen in the second-order discretization

$$\frac{T^{k+1}_{i-1,j} - 2T^{k+1}_{i,j} + T^{k}_{i+1,j}}{\Delta x^2} + \frac{T^{k+1}_{i,j-1} - 2T^{k+1}_{i,j} + T^{k}_{i,j-1}}{\Delta y^2} = S^k(x, y) \tag{20}$$

for Equation (12). However, this restriction is relaxed on the subdomain boundaries by providing $T^k_{i-1,j}$ and $T^k_{i,j-1}$ rather than $T^{k+1}_{i-1,j}$ and $T^{k+1}_{i,j-1}$. Thus, the parallel algorithm is termed a Gauss-Seidel/Block Jacobi scheme, signifying that the interior points are handled via a Gauss-Seidel scheme, and some of the interfacial points are handled via a Jacobi scheme. In this work, the SSOR scheme is employed with $\omega = 0.98$, and the solution is considered converged when the infinity norm of the matrix is less than 10^{-3}.

C. Use of Offload Mode on the Intel MIC Architecture

Offload mode is a very similar operation to the current paradigm offered by GPGPUs. The serial portion of the code is run on the CPU, and portions of the code are either explicitly or implicitly migrated to the Intel MIC card for execution. In this code, implicit offloading is used due to structures needed in offload. This is accomplished by using directives such as `#pragma offload_attribute( push, _Shared)` and `#pragma offload_attribute(pop)` at the beginnings and ends of sections to be offloaded. This paradigm allows for simple conversion of hybrid MPI and

TABLE I: Speed Up for the Poisson Solver Offloaded to Intel MIC

Number of Threads	Speed Up (600x600)	Speed Up (1200x1200)	Speed Up (2400x2400)
1	1.0000	1.0000	1.0000
15	5.3398	9.5367	14.9551
30	7.5597	14.1972	24.0569
60	8.3855	16.1706	29.9020
90	9.6261	18.6117	35.3985
120	1.2146	23.9031	45.6310

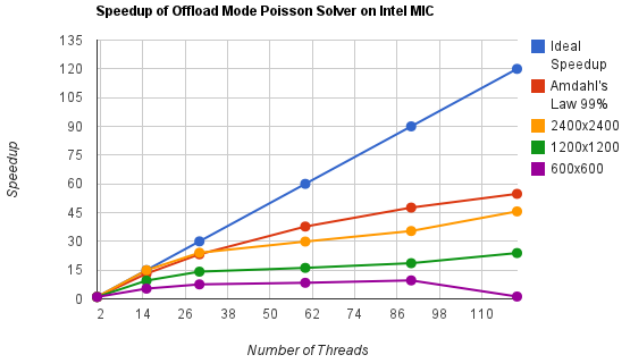


Fig. 15: Strong Scaling Plot for Three Matrix Sizes

OpenMP codes to MIC-capable codes in the same way that OpenACC allows conversion to GPGPU-capable codes.

D. Parallel Performance Results: Strong Scaling Study on the Intel MIC

In this work, the Poisson solver is run on one core of an Intel Xeon 5600 processor, and the Gauss-Seidel solver loop is offloaded to 1, 15, 30, 60, 90, and 120 threads on one Intel KNF card. Speedup improves as the size of the matrix being solved grows, as can be seen in Figure 15, in which the baseline speed is that observed for each case when offloading to only one thread. Also, the code performs well in comparison to Amdahl's law [17] for a serial fraction of 99%.

Table I and Figure 15 indicate that there is not enough work in the 600x600 case to effectively use more than a small amount the computational capability of the Intel KNF card and that it would be necessary to grow the matrix considerably to take full advantage of the Intel KNF in offload mode for this problem.

VI. Conclusion

The Intel MIC architecture offers unique advantages that make it an appealing choice for sustainable software development for scientific computing. First and foremost, it offers a standards-based programming model that maintains source code compatibility with existing Xeon (and other x86-based) architectures. Additionally, the MIC is expected to employ a unified development environment (compiler, debugger, and performance tools) that is applicable to all Intel architectures, and the unified compiler is expected to produce binaries that run appropriately on any Intel architecture. Aside from providing fexibility and ease-of-use, the Intel MIC offers easily accessible scalability across its many cores, as indicated by the scaling results presented for the initial porting efforts of the applications studied in this work. Based on the work presented here (and the results of other experiments planned for future publication), it appears that codes exhibiting sufficient medium-grain parallelism scale effectively across all of the cores on the Intel MIC. This scaling behavior coupled with Intel's demonstration of a pre-release Intel Knights Corner accelerator performing at 1 TFLOP per second at SC11 [18] suggests that significant sustained performance is expected for optimized codes that exhibit sufficient fine-grain and medium-grain parallelism. For these reasons and those discussed above, the Intel MIC is expected to be a highly productive platform for scientific computing.

VII. Acknowledgement

The authors thank Intel, the Joint Institute for Computational Sciences of the University of Tennessee, and the National Institute for Computational Sciences (NICS) for making this research possible. Special thanks are extended to Rob Van der Wijngaart of Intel for optimizing the BGK model Boltzmann solver used in this work.

References

[1] Z. H. Li and H. X. Zhang, "Study on gas kinetic unified algorithm for flows from rarefied transition to continuum," *Journal of Computational Physics*, vol. 193, pp. 708–738, 2004.

[2] R. G. Brook, "A parallel, matrix-free newton method for solving approximate boltzmann equations on unstructured topologies," Ph.D. dissertation, University of Tennessee at Chattanooga, 2008.

[3] H. H. Cheng, "Programming with dual numbers and its applications in mechanisms design," *Engineering with Computers*, vol. 10, no. 4, pp. 212–229, 1994.

[4] G. A. Sod, "A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws," *Journal of Computational Physics*, vol. 27, pp. 1–31, 1978.

[5] L. Mieussens, "Discrete velocity model and implicit scheme for the bgk equation of rarefied gas dynamics," *Mathematical Models and Methods in Applied Sciences*, vol. 10, pp. 1121–1149, 2000.

[6] A. J. Chorin, "The numerical solution on the navier-stokes equations for an incompressible fluid," *American Mathematical Society*, vol. 73, p. 928, 1967.

[7] B. Smith, P. Bjorstad, and W. Gropp, *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[8] M. Garbey and D. T. Dervout, "On some aitken like acceleration of the schwarz method," *International Journal for Numerical Methods in Fluids*, vol. 40, pp. 1493–1513, 2002.

[9] M. Garbey, "Acceleration of the schwarz method for elliptic problems," *Journal on Scientific Computing*, vol. 26, no. 6, pp. 1871–1893, 2005.
[10] B. Hadri, "Efficient parallel computation to simulate blood flow," Ph.D. dissertation, University of Houston, 2008.
[11] B. Hadri and M. Garbey, "A fast navier-stokes flow simulation tool for image based cfd," *Journal of Algorithms and Computational Technology*, vol. 2, no. 24, pp. 527–556, 2008.
[12] P. Angot, C. Bruneau, and P. Fabrie, "A penalisation method to take into account obstacles in viscous flows," *Numerische Mathematik*, vol. 81, pp. 497–520, 1999.
[13] E. Arquis and J. Caltagirone, "Sur les conditions hydrodynamiques au voisinage d'une interface milieu fluide-milieux poreux: Application a la convection naturelle," *CRAS, Paris II*, vol. 299, pp. 1–4, 1984.
[14] T. Chan and L. Vese, "Active contours without edges," *IEEE Transaction on Image Processing*, vol. 10, no. 2, pp. 266–277, 2001.
[15] C. S. Peskin, "The immersed boundary method," *Acta Numerica*, pp. 1–39, 2002.
[16] "Visit visualization software," "http://www.llnl.gov/visit".
[17] G. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," in *AFIPS Conference Proceedings*, vol. 30, 1967, pp. 483–485.
[18] P. Darling, "First intel many integrated core co-processor demonstrated to deliver performance above 1 tflops," http://newsroom.intel.com/community/intel_newsroom/blog/2011/11/15/intel-reveals-details-of-next-generation-high-performance-computing-platforms, November 2011, accessed April 30, 2012.