Scalability of 3D deterministic particle transport on the Intel MIC architecture Postprint

Authors: WANG Qing-Lin, LIU Jie, GONG Chun-Ye, XING Zuo-Cheng

Date: 2023-06-18T00:00:00+00:00

Abstract

The key to large-scale parallel solutions of deterministic particle transport problem is single-node computation performance. Hence, single-node computation is often parallelized on multi-core or many-core computer architectures. However, the number of on-chip cores grows quickly with the scale-down of feature size in semiconductor technology. In this paper, we present a scalability investigation of one energy group time-independent deterministic discrete ordinates neutron transport in 3D Cartesian geometry (Sweep3D) on Intel' s Many Integrated Core (MIC) architecture, which can provide up to 62 cores with four hardware threads per core now and will own up to 72 in the future. The parallel programming model, OpenMP, and vector intrinsic functions are used to exploit thread parallelism and vector parallelism for the discrete ordinates method, respectively. The results on a 57-core MIC coprocessor show that the implementation of Sweep3D on MIC has good scalability in performance. In addition, the application of the Roofline model to assess the implementation and performance comparison between MIC and Tesla K20C Graphics Processing Unit (GPU) are also reported.

Full Text

Preamble

Scalability of 3D Deterministic Particle Transport on the Intel MIC Architecture

WANG Qing-Lin (王庆林),[1],† LIU Jie (刘杰),[1] GONG Chun-Ye (龚春叶),[2] and XING Zuo-Cheng (邢座程)[1]
[1]Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha 410073, China
[2]Science and Technology on Space Physics Laboratory, Beijing 100076, China

(Received November 19, 2014; accepted in revised form January 5, 2015; published online October 20, 2015)

The key to large-scale parallel solutions of deterministic particle transport problems is single-node computation performance. Hence, single-node computation is often parallelized on multi-core or many-core computer architectures. However, the number of on-chip cores grows quickly with the scale-down of feature size in semiconductor technology. In this paper, we present a scalability investigation of one energy group time-independent deterministic discrete ordinates neutron transport in 3D Cartesian geometry (Sweep3D) on Intel' s Many Integrated Core (MIC) architecture, which can provide up to 62 cores with four hardware threads per core now and will own up to 72 in the future. The parallel programming model, OpenMP, and vector intrinsic functions are used to exploit thread parallelism and vector parallelism for the discrete ordinates method, respectively. The results on a 57-core MIC coprocessor show that the implementation of Sweep3D on MIC has good scalability in performance. In addition, the application of the Roofline model to assess the implementation and performance comparison between MIC and Tesla K20C Graphics Processing Unit (GPU) are also reported.

Keywords: Particle transport, Discrete ordinates method, Sweep3D, Many Integrated Core (MIC), Scalability, Roofline model, Graphics Processing Unit (GPU)

DOI: 10.13538/j.1001-8042/nst.26.050502

Introduction

The discrete ordinates (Sn) method is one of the most popular deterministic numerical methods for solving particle transport equations [**?**]. The particle transport equation depends on the energy, angular directions, and spatial coordinates of the particles with an underlying medium so that the solution is highly computation-intensive [**?**]. Some works \cite{3–5} reveal that single-node computation speed is the key performance factor, rather than inter-processor communication performance, in the discrete ordinates method for large-scale problems. The advent of on-chip multi-core or many-core computer architectures makes it possible to overcome these challenges.

The Sweep3D benchmark [**?**] is a time-independent, single-group, discrete ordinates 3D Cartesian geometry deterministic neutron transport code, which is extracted from real Department of Energy (DOE) Accelerated Strategic Computing Initiative (ASCI) applications. It represents the heart of the discrete ordinates method for solving particle transport equations. Recently, Sweep3D has been accelerated on different multi-core or many-core computer architectures \cite{7–13}.

Petrini et al. [**?**] ported Sweep3D to the Cell Broadband Engine (B.E.) processor with an on-chip single-MPI routine and achieved good performance by exploit-

ing different levels of parallelism and data streaming in Sweep3D. In contrast, Lubeck et al. [?] developed a library of on-chip SPU-to-SPU communication routines and utilized on-chip multi-MPI routines to accomplish an SPU-centric implementation of Sweep3D on Cell B.E., achieving better performance due to reduced data movement. With the application of many-core architectures in high-performance computing, Sweep3D has been implemented on Graphics Processing Unit (GPU) [?, ?] and Intel Many Integrated Core (MIC) [?]. Gong et al. [?] used multi-dimensional optimizations to parallelize Sweep3D on GPU and reached a speedup of 2.25 times compared to the CPU-based version when flux fixup was disabled. Moreover, more concurrent threads were obtained by extracting parallelism in the data-dependent loop of solving recursive Sn equations in Sweep3D, leading to further performance improvement on GPU [?]. Wang et al. [?] applied both hardware threads and vector units in MIC to efficiently exploit multi-level parallelism in Sweep3D and achieved a 1.99 times speedup based on an eight-core Intel Xeon E5-2660 CPU.

The number of transistors integrated on a single chip increases exponentially as the semiconductor industry continues to scale down feature size. This growth permits the rapid development of Chip Multi-processor (CMP) architectures. One evolving trend of CMP architectures is integrating more general-purpose cores on each chip, such as the Intel MIC architecture [?], which is built on lightweight x86 cores. MIC was introduced as a highly parallel coprocessor by Intel Corporation in 2010. The earlier prototype card of the Intel MIC architecture, codenamed Knights Ferry (KNF), provides up to 32 cores with four threads per core in a single chip. The first-generation product, codenamed Knights Corner (KNC), consists of up to 62 cores per chip. The second-generation product, codenamed Knights Landing (KNL), will feature up to 72 cores with four threads per core in the future [?]. Therefore, it is important to study how parallel algorithms scale with the number of cores on CMP architectures. Wang et al. [?] studied the scalability of embarrassingly parallel algorithms on MIC, while Saule et al. [?] evaluated the scalability of three graph algorithms on MIC. However, there is little work on scalability evaluation of 3D deterministic particle transport on the Intel MIC architecture.

This work builds upon our previous research on parallelizing 3D deterministic particle transport on MIC [?]. We present an investigation of the scalability of 3D deterministic particle transport algorithms on MIC. Our results on a 57-core MIC chip show that the algorithm can scale well with the rapidly increasing number of cores in MIC. Without flux fixup, the algorithm' s performance is determined by the number of cores and the off-chip memory bandwidth in the MIC. With flux fixup, most of the algorithm cannot be vectorized, so performance depends mainly on the number of cores in the MIC. Additionally, we apply the Roofline model to assess the implementation of Sweep3D on MIC and compare it with the GPU implementation running on Tesla K20C GPU.

The structure of this paper is as follows. Section II presents the relevant background. Section III describes the MIC-based Sweep3D implementation in detail.

Performance and scalability results are collected in Sec. IV. Sec. V concludes and describes future work.

II. Background

A. Sweep3D

The time-independent single-group particle transport equation is shown in Eq. (1). The unknown field is $\Psi(\mathbf{r},\ )$, which represents the flux of particles traveling in direction at spatial point $\mathbf{r}$. σ_t is the total cross-section and σ_s is the scattering cross-section from $'$ into at $\mathbf{r}$. The right-hand side of the equation is the source term, including the scattering source and external source. $Q_{ext}(\mathbf{r},\ )$ expresses the external source.

$$\cdot \nabla\Psi(\mathbf{r},\ ) + \sigma_t(\mathbf{r})\Psi(\mathbf{r},\ ) = \int \sigma_s(\mathbf{r},\ ' \rightarrow\ )\Psi(\mathbf{r},\ ')d\ ' + Q_{ext}(\mathbf{r},\ ).$$

In Sweep3D, the angular direction is discretized into a set of quadrature points and the space is divided into a finite mesh of cells. The XYZ geometry is represented by an IJK logically rectangular grid of cells. Source Iteration (SI) scheme is used to handle angular couplings by scattering media. Each iteration includes computing the iterative source, wavefront sweeping, computing flux error, and judging whether the convergence condition is met. Wavefront sweeping is the most time-consuming part. In Cartesian geometry, each octant of angle sweeps has a different sweep direction through the physical space, and all angles in a given octant sweep the same way.

Integrating the left-hand and right-hand sides of Eq. (1) over the neighboring angular-directions region, $\Delta\ _m$, of a given discrete angle, $\ _m(\mu_m, \eta_m, \xi_m)$, we obtain the balanced equation as follows:

$$\mu_m \frac{\partial \Psi_m}{\partial x} + \eta_m \frac{\partial \Psi_m}{\partial y} + \xi_m \frac{\partial \Psi_m}{\partial z} + \sigma_t(\mathbf{r})\Psi_m = Q_m(\mathbf{r}).$$

In the SI scheme, $Q_m(\mathbf{r})$ is known. The diamond space difference scheme is applied to obtain three auxiliary equations so that each grid cell has 4 equations (3 auxiliary plus 1 balance) with 7 unknowns (6 faces plus 1 central). Boundary conditions initialize the sweep and allow the system of equations to be completed. For any given cell, three known inflows enable the cell center and three outflows to be obtained, and then the three outflows provide inflows to three adjoining cells in particle traveling directions. Therefore, there is a recursion dependence in all three grid directions. This recursion dependence causes the sweep to proceed in a diagonal wave across the physical space, shown in Fig. 1 [Figure 1: see original paper]. Consequently, parallelism is limited by the length of the JK-diagonal line. To alleviate this problem, MMI angles for each octant are pipelined on JK-diagonal lines to increase the number of parallel I-lines.

Moreover, Sweep3D utilizes Diffusion Synthetic Acceleration (DSA) [**?**] to improve its convergence of the source iteration scheme. The wavefront sweeping subroutine mainly involves computing sources from spherical harmonic (Pn) moments, solving Sn equations recursively with or without flux fixup, updating flux from Pn moments, and updating DSA face currents, as shown in Fig. 2 [Figure 2: see original paper].

B. MIC Architecture

The MIC architecture is shown in Fig. 3 [Figure 3: see original paper]. Each MIC chip consists of many general-purpose cores. These cores are in-order and issue two instructions per cycle to two asymmetrical pipelines (u- and v-pipes). Both pipelines can handle scalar operations, whereas only the u-pipe can execute vector operations. One vector unit in MIC can perform 16 single-precision or 8 double-precision floating-point operations simultaneously. The vector units also support floating-point fused multiply-add operations. Each core maintains four hardware threads and schedules these threads by round-robin. Each core also has a 32-KB private L1 instruction cache, 32-KB private L1 data cache, and 512-KB unified L2 cache. The two-level caches of all cores are kept coherent by a distributed tag directory. There are multiple on-die Memory Controllers (MCs), which connect off-chip high-bandwidth Graphics Double-Data Rate, version 5 memory (GDDR5) to on-chip cores and L2 caches in the MIC. All cores, MCs, and caches on the chip communicate via a high-bandwidth bidirectional ring bus.

The MIC architecture supports many parallel programming models, such as Open Multi-Processing (OpenMP), Intel Cilk Plus, and Intel Threading Building Blocks (TBB). In our implementation, we use OpenMP to exploit the computing capability of parallel hardware threads in MIC. The ways to utilize the vector units in MIC include compiler automatic vectorization and vector intrinsic functions. As compiler automatic vectorization is only possible for simple loops, we apply intrinsic functions to exploit vector parallelism in the discrete ordinates method. Additionally, two usage models are available in MIC: offload and native. In the offload model, MIC runs as a coprocessor like GPU. In the native model, MIC acts as a many-core processor. In contrast with the usage model of GPU, the native model is unique to MIC. To eliminate the influence of communication between the CPU host and MIC coprocessor, we use the native model of MIC to evaluate the scalability of 3D deterministic particle transport on MIC.

III. Details of MIC-Based Implementation

To achieve full utilization of parallel computing resources in MIC, a MIC-based particle transport solver is proposed, as shown in Fig. 4 [Figure 4: see original paper]. In the solver, we use both thread parallelism and vector parallelism to exploit parallelism in each procedure of the discrete ordinates method.

A. Thread Parallelism

As shown in Fig. 4, all three procedures of Sweep3D are processed with many parallel threads, while thread parallelism in these procedures differs.

The iterative source equals iterative scattering moments plus the external source, shown in Eq. (3):

$$S_l(\mathbf{r})^i = \sigma_s(\mathbf{r})\Phi_l(\mathbf{r})^{i-1} + [Q_{ext}(\mathbf{r})]_{l==0},$$

where i represents the ith iteration loop. There is no dependence among the calculations of iterative source for all cells, so the iterative source of all cells in each iteration can be computed in parallel. The OpenMP primitive `parallel_{for}` can be utilized to schedule parallel tasks to all hardware threads. The granularity of scheduling tasks can be one I-line grid column or one k-plane grid block. In Sweep3D, the number of k-plane grid blocks may be too small to occupy all hardware threads. Hence, one I-line grid column is set as the minimum granularity for partitioning tasks in thread-level parallelization of computing iterative source.

As one JK-diagonal line provides input to the adjoining JK-diagonal line in the particle traveling direction, it is very difficult to extract thread-level parallelism in wavefront sweeping. Fortunately, all I-lines in each JK-diagonal line are independent of each other. The parallelism is limited by the number of I-lines. To increase parallelism, MMI angles for each octant are pipelined. As shown in Fig. 1, when the problem sizes of the I and K dimensions are both six and three different angles are processed in pipeline, the number of parallel I-lines for a given diagonal increases from five to twelve. The OpenMP primitive `parallel_{for}` is added to parallelize all I-lines of all JK-diagonal lines for a given diagonal value, corresponding to Line 7 in Fig. 2. In the implementation, MMI equals the number of discrete angles per octant.

The maximum relative flux error is computed across all cells, as shown in Eq. (4):

$$Error_{max} = \max \left| \frac{\Phi(\mathbf{r})^i - \Phi(\mathbf{r})^{i-1}}{\Phi(\mathbf{r})^i} \right|.$$

Like computing the iterative source, there is no dependence among the computations of relative flux error for all cells. The OpenMP primitive `parallel_{for}` is also applied to compute the maximum relative flux error in cells of some I-line grid columns in parallel, with the reduction function directive `max` used to obtain the maximum among all OpenMP threads.

B. Vector Parallelism

In thread-level parallelization, the granularity of distributing tasks is one I-line grid column. Therefore, vector units can be used to exploit parallelism within one I-line grid column, as shown in Fig. 4. Except for computing relative flux error and solving recursive Sn equations, all other procedures of Sweep3D are easily vectorized due to independence among calculations of all cells in one I-line grid column. Although the computations of relative flux error for all cells are also independent, they involve division operations and branches generated by zero-value judgments. Thus, computing relative flux error is processed by scalar units rather than vector units in MIC. Below, we mainly describe the implementation of exploiting vector parallelism in solving recursive Sn equations.

For solving recursive Sn equations, the balanced equation shown in Eq. (2) is transformed to Eq. (5) for one I-line grid column, with auxiliary equations shown in Eq. (6):

$$\Phi_n(\mathbf{r},\Omega_m) = \frac{1}{D_i}[Q_m + C_i\Phi_n(I_i,\Omega_m)^i + C_j\Phi_n(J_i,\Omega_m)^i + C_k\Phi_n(K_i,\Omega_m)^i],$$

$$\Phi_n(\mathbf{r},\Omega_m)^o_{I,J,K} = 2\Phi_n(\mathbf{r},\Omega_m) - \Phi_n(\mathbf{r},\Omega_m)^i_{I,J,K},$$

where $\Phi_n(I_i,\Omega_m)^i$, $\Phi_n(J_i,\Omega_m)^i$, and $\Phi_n(K_i,\Omega_m)^i$ are the input fluxes of the (i,j,k) cell in the I, J, K directions, respectively; $\Phi_n(\mathbf{r},\Omega_m)^o_{I,J,K}$ is the output flux in the I, J, K direction of the (i,j,k) cell; $\Phi_n(\mathbf{r},\Omega_m)$ is the central flux of the cell for the current discrete angle; and D_i, C_i, C_j, and C_k represent relative difference parameters. The central flux $\Phi_n(\mathbf{r},\Omega_m)$ cannot be negative when the input fluxes $\Phi_n(\mathbf{r},\Omega_m)^i_{I,J,K}$ are non-negative in Eq. (5), while the output fluxes $\Phi_n(\mathbf{r},\Omega_m)^o_{I,J,K}$ obtained through Eq. (6) may be negative. We can choose whether negative fluxes should be fixed in the sub-procedure. If flux fixup is enabled, solving recursive Sn equations becomes full of judgments and branches, making it hard to vectorize. If flux fixup is disabled, data dependence remains in solving recursive Sn equations in I-line grid columns, but we can use loop unrolling and splitting to parallelize the sub-procedure.

The parallelization of the simplified Sn recursion without flux fixup is shown in Algorithm 1. The arrays B and C store $\Phi_n(\mathbf{r},\Omega_m)^i$ and $\Phi_n(\mathbf{r},\Omega_m)^o$ in the J and K directions. The array phi stores $\Phi_n(\mathbf{r},\Omega_m)$. `it` represents the problem size in the I dimension. `phiir` stores the output and input flux in the I direction of the (i,j,k) cell and is the dependence variable in the sub-procedure. Thus, we can isolate the computations (Lines 6–8) relating to `phiir`, while the remaining parts (Lines 3–5 and 9–11) for cells in one I-line grid column can be completed in parallel. The loop unrolling factor is set to eight to maintain good data locality. Compared to serial implementation, only two memory store and one load operation are added, while most operations are processed in parallel.

The implementation of Lines 9–11 in Algorithm 1 using vector intrinsic functions on MIC is shown in Lines 14–18 of Algorithm 2. Further, software prefetch operations (Lines 8–9) are inserted to improve memory access cost. All vectorized procedures of the MIC-based Sweep3D in Fig. 4 are implemented using intrinsic functions.

IV. Performance Results and Discussion

A. Experimental Setup

The experimental platform contains an eight-core Intel Xeon E5-2660 CPU with 128GB of memory, a MIC coprocessor, and a GPU coprocessor. The target MIC coprocessor includes 57 cores running the Intel MIC software stack. A C and Fortran hybrid version of Sweep3D is implemented on MIC and compiled by the Intel C and Fortran compiler with level-three optimization. To compare MIC to GPU, the GPU version of Sweep3D from Ref. [**?**] is used and compiled by the nvcc compiler with level-three optimization. Both codes run in double-precision floating-point. The specifications are described in Table 1 .

Table 1. Specification of the experiment platform

Component	Specifications
CPU	Intel Xeon E5-2660, 8 cores, 2.2 GHz
MIC	57 cores, 4 threads/core, 1.1 GHz, 5 GB GDDR5
GPU	Tesla K20C, 2496 CUDA Cores, 0.71 GHz, Capability 3.5, 5 GB memory
Operating System	Linux Red Hat 4.4.5-6
Intel Compiler	Intel v13.0.0, ifort, icc, OpenMP v3.1
Nvcc Compiler	GCC v4.4.6, NVCC v6.0.1

In the simulation, we consider vacuum boundary conditions, which correspond to zero incoming flux in the domain, and DSA face-current calculations. The Pn scattering order is set to 1, and center (1/3)-cubed grid points have a unit source in 3D geometry. The number of source iterations is set to four in Sweep3D. We run each instance of Sweep3D ten times and take the shortest runtime as the measurement.

B. Performance under Different Optimizations

To achieve the best performance, the affinity type between OpenMP threads and hardware cores is set to scatter mode, allowing OpenMP threads to be allocated to as many hardware cores as possible in MIC.

The execution times of different optimizations without flux fixup are shown in Fig. 5 [Figure 5: see original paper]. As total memory usage increases with the

number of spatial cells, the maximum problem size is set to 320^3, which requires 4201 MB of memory. When the problem size is 128^3, vectorization reduces execution time by 29% compared to implementation exploiting only thread-level parallelism. When the problem size increases to 256^3, runtime reduction reaches up to 50%. Even at the maximum problem size, a runtime reduction of about 44% is obtained. Therefore, vectorization is important for parallelizing Sweep3D without flux fixup on MIC. However, the performance improvement from vectorization is far less than the number of operations one vector unit can handle simultaneously. The main reason is that performance depends primarily on MIC' s memory performance due to the small ratio between floating-point operations and memory loads in Sweep3D. In other words, the algorithm' s performance without flux fixup highly relies on off-chip memory bandwidth in MIC.

Figure 6 [Figure 6: see original paper] shows performance comparisons of Sweep3D with negative flux fixup between different optimizations. When the problem size is 128^3, vectorization optimization reduces execution time by only 10%. As problem size grows, runtime decrease is at most 25%. Compared to runtime reduction by vectorization without flux fixup, the decrease with flux fixup is much smaller. Two main reasons explain this phenomenon. First, the sub-procedure of solving recursive Sn equations with flux fixup is full of judgments and branches, making it hard to process with vector units. Second, the proportion of runtime spent solving recursive Sn equations in total execution time with flux fixup is higher than without flux fixup. Therefore, exploiting thread parallelism rather than vector parallelism is key to achieving high performance from the discrete ordinates method with flux fixup on MIC.

To assess absolute performance of different optimizations, we apply the Roofline model [**?**] to the target MIC (Fig. 7 [Figure 7: see original paper]). The Roofline model measures the maximum performance an algorithm can achieve on a given hardware architecture and can compare performance across different systems. The model includes three basic factors: peak floating-point performance (GFlops/s), peak memory bandwidth (GBytes/s), and operational intensity (Flops/Byte). Peak performance and memory bandwidth depend on hardware. Two horizontal lines in Fig. 7 show peak FLOPS under different optimizations, determined by MIC hardware specifications. Peak memory bandwidth is measured with the STREAM benchmark [**?**]. Using the STREAM-BIG-C test, we obtain a peak memory bandwidth of 80.62 Gbytes/s. The slanted line exhibits maximum algorithm performance limited by memory bandwidth. Operational Intensity (OI) means FLOPS per byte of traffic between caches and memory, determined by algorithms. In Sweep3D, the wavefront sweeping subroutine often takes over 90% of total runtime, so we only consider this subroutine when comparing theoretical and experimental performance. Without flux fixup, operational intensity is constant for any problem size and source iteration step. With flux fixup, operational intensity increases with the number of flux fixups in the subroutine, resulting in different intensities across source iterations.

Table 2 compares theoretical and experimental performance under different optimizations. Without flux fixup, operational intensity is about 0.3583 Flops/Byte. With flux fixup, minimum and maximum operational intensities in four iterations are 0.3616 Flops/Byte and 0.3811 Flops/Byte, respectively. As seen in Fig. 7, theoretical performance equals memory bandwidth times operational intensity due to low operational intensities. Experimental results are obtained by measuring execution time of the wavefront sweeping subroutine. Without flux fixup, theoretical performance is 28.89 Gflops/s and maximum experimental performance is 21.04 Gflops/s. With flux fixup, theoretical performance increases with operational intensity, but experimental performance with maximum operational intensity is less than with minimum intensity under the same optimizations. The chief reason is that flux fixup worsens load balance between OpenMP threads. With flux fixup, minimum theoretical performance is 29.15 Gflops/s and corresponding maximum experimental performance is 14.94 Gflops/s. All experimental performances are lower than theoretical, with maximum efficiency about 72.8%. Two factors cause this gap. First, varying thread parallelism along sweep directions leads to load imbalance among hardware threads, and exploited vector parallelism is insufficient to fill the SIMD pipeline of each core. Second, the Roofline model is imperfect. Obtainable peak memory bandwidth is influenced by the number of threads, which depends on varying thread parallelism. Additionally, synchronization and scheduling overhead of many OpenMP threads cannot be omitted and rises quickly with thread count, but neither memory bandwidth variance nor multi-threading overhead is included in the Roofline model, making theoretical performance appear high.

Table 2. Comparison between predicted and experimental performance of the wavefront sweeping subroutine on MIC under different optimizations (problem size 320^3, performance in Gflops/s)

Flux fixup	OI (Flops/Byte)	Multi-thread	Multi-thread + Vectorization
Off	0.3583	Pred. 28.89, Expt. 18.45	Pred. 28.89, Expt. 21.04
On (Min)	0.3616	Pred. 29.15, Expt. 13.05	Pred. 29.15, Expt. 14.94
On (Max)	0.3811	Pred. 30.73, Expt. 11.89	Pred. 30.73, Expt. 13.71

C. Performance Comparison between MIC and GPU

Figure 8 [Figure 8: see original paper] shows the speedup of Sweep3D on MIC compared to GPU under various problem sizes. The double-precision floating-point peak performance ratio of MIC to GPU is about 0.85:1.00. Speedup gradually increases with problem size, regardless of flux fixup status.

Without flux fixup, MIC is inferior to GPU when problem size is less than

288^3. At 128^3, the MIC-to-GPU performance ratio is only 0.58:1.00. Only when problem size reaches 192^3 does the performance ratio roughly equal the peak performance ratio. This phenomenon mainly results from insufficient core utilization. Hardware cores in MIC act like Streaming Multiprocessors (SMs) in GPU. The target MIC has 57 cores, while the target GPU has only 13 SMs. Therefore, core utilization in MIC is worse than in GPU for small problem sizes. When problem size is greater than or equal to 288^3, MIC performance matches GPU.

With flux fixup enabled, MIC is superior to GPU for all examined problem sizes. At the smallest problem size 128^3, MIC achieves 1.13 times speedup over GPU. At the largest problem size 320^3, speedup reaches 1.67 times. The GPU architecture consists of simple cores, while MIC is based on general-purpose cores. Thus, MIC is much better at processing judgment expressions than GPU. As stated in Sec. IV.B, solving recursive Sn equations with flux fixup includes many judgment expressions and is hard to efficiently parallelize with vector units in MIC as well as SIMT units in GPU. Therefore, much better performance is achieved on MIC than GPU with flux fixup.

D. Scalability with the Number of Cores

To effectively evaluate performance variance with core count, OpenMP threads are bound to specific hardware cores in scalability evaluation. Two primary ways to scale Sweep3D on MIC are strong scaling and weak scaling. Strong scaling means applying more cores to the same problem size for faster results. Weak scaling refers to increasing problem size as Sweep3D runs on more cores.

1. Strong Scaling Evaluation Figure 9 [Figure 9: see original paper] shows execution times of Sweep3D without flux fixup running on different numbers of MIC cores when problem size is 256^3. The horizontal axis represents the number of hardware threads, which equals cores times hardware threads per core. One core consists of four hardware threads in MIC. To fully utilize these threads, four OpenMP threads must be assigned per core. Using all 57 cores in the target MIC provides 228 parallel hardware threads, requiring 228 OpenMP threads. With only one core, Sweep3D runtime is 170.08 seconds. Two cores reduce runtime to 86.23 seconds, yielding a speedup of about 1.97 times. When core count increases to 32, runtime drops to 8.61 seconds, achieving 19.76 times speedup from 1 to 32 cores. When all 57 cores are utilized, runtime is 6.64 seconds, achieving 25.6-fold speedup from 1 to 57 cores. Therefore, runtime continues to diminish with more cores.

Figure 10 [Figure 10: see original paper] shows performance of Sweep3D with negative flux fixup running on different numbers of MIC cores when problem size is 256^3. As stated in Sec. IV.B, achieving high performance with flux fixup on MIC lies in exploiting thread parallelism rather than vector parallelism. Therefore, much larger speedup is achieved when more cores or hardware threads

are employed. Speedup from 1 to 32 cores is about 22.78 times, while it reaches 34.59 times from 1 to 57 cores.

Figure 11 [Figure 11: see original paper] exhibits speedup from 1 to 57 cores under different problem sizes. Without flux fixup, speedup rises gradually with problem size initially. At 256^3, maximum speedup of 25.61 times is obtained. Under larger problem sizes, a plateau is reached and even tiny drops are observed. This occurs because simulation without flux fixup is limited by peak memory bandwidth, so more parallelism with larger problem sizes cannot be efficiently exploited by parallel cores. Hence, maximum strong scaling efficiency is about 45% without flux fixup. With flux fixup, speedup increases continuously with problem size. At 320^3, speedup from 1 to 57 cores achieves 35.90 times. Therefore, maximum strong scaling efficiency can reach 63% with flux fixup. However, strong scaling efficiency is not high in all cases. Runtime is chiefly determined by wavefront sweeping in Sweep3D. The number of parallel I-lines in wavefront sweeping first increases from one and then decreases to one with particle flowing, shown in Fig. 1. Thus, load imbalance occurs among cores when more than one core is utilized. Using more cores further aggravates load imbalance, reducing core utilization efficiency. Consequently, obtained strong scaling efficiency is not high.

2. Weak Scaling Evaluation Table 3 shows execution times of Sweep3D running on different numbers of MIC cores. For problem sizes, I and J dimensions are fixed to 256, while K dimension maintains a linear relationship with the number of utilized cores. When core count is less than or equal to 16, runtime exhibits small variations with problem size. When more than 16 cores are used, runtime changes greatly with problem size. A chief cause is again load imbalance from changing parallelism in sweeping. Load imbalance sharply worsens under more than 16 cores and corresponding problem sizes, causing rapid runtime increases. Without flux fixup, runtime under one core is 7.25 seconds and rises to 11.59 seconds finally, yielding weak scaling efficiency of about 63%. With flux fixup, execution time is 14.66 seconds initially and grows to 17.59 seconds at the end, reaching 83% weak scaling efficiency.

Table 3. Execution times of Sweep3D when problem sizes increase linearly with utilized cores in MIC

No. of core × hardware thread/core	Problem size	Runtime (s) Without flux fixup	Runtime (s) With flux fixup
1 × 4	256$×256×$8	25	14.66
2 × 4	256$×256×$16	51	14.71
4 × 4	256$×256×$32	28	14.79
8 × 4	256$×256×$64	2	15.02
16 × 4	256$×256×$128	8	15.58
32 × 4	256$×256×$256	6	16.45
57 × 4	256$×256×$456	59	17.59

Overall, these results show our implementation has good scalability on MIC, regardless of flux fixup status, and can achieve good performance with more cores.

E. Comparison with Previous Work

The difference between the new implementation in this paper and our old one [**?**] is utilization of software prefetch technology to improve memory access cost in vectorization. For problem size 256^3, the new and old implementations cost 6.64 and 6.78 seconds, respectively, with flux fixup disabled and all 57 cores used. When only one MIC core is utilized, execution times are 170.08 and 187.37 seconds, respectively. With flux fixup, the new implementation costs 10.32 seconds on all cores and 356.98 seconds on one core, while the old demands 10.41 and 370.49 seconds. Therefore, minimum and maximum performance improvements over the old implementation are 0.93% and 10.17%, respectively. Our new implementation thus shows better performance portability across core counts in MIC.

V. Conclusion and Future Work

In this paper, the parallel programming model OpenMP and vector intrinsic functions are used to implement Sweep3D parallelization on MIC. All three procedures of Sweep3D are processed in parallel. Except for computing relative flux error and solving recursive Sn equations with flux fixup, all other sub-procedures are vectorized on MIC. We evaluate the influence of different parallel optimizations on performance and apply the Roofline model to assess absolute performance. Results show that algorithm performance is determined by core count and off-chip memory bandwidth in MIC without flux fixup, while speed depends mainly on core count with flux fixup. Investigation of MIC-based Sweep3D scalability demonstrates good performance scalability with the Intel MIC architecture. Moreover, comparison between our MIC implementation and prior Tesla K20C GPU implementation is discussed.

In the future, performance and scalability issues of Sweep3D on heterogeneous CPU/MIC clusters like Tianhe-2 supercomputer will be studied.

References

[1] Colomer G, Borrell R, Trias F X, et al. Parallel algorithms for Sn transport sweeps on unstructured meshes. J Comput Phys, 2013, 232: 118–135. DOI: 10.1016/j.jcp.2012.07.009

[2] Marchuk G I and Lebedev V I. Numerical methods in the theory of neutron transport. New York (USA): Harwood Academic Publishers, 1986, 3–26.

[3] Hoisie A, Lubeck O, Wasserman H, et al. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional

wavefront applications. Int J High Perform C, 2000, 14: 330-346. DOI: 10.1177/109434200001400405

[4] Fischer J W and Azmy Y Y. Comparison via parallel performance models of angular and spatial domain decompositions for solving neutral particle transport problems. Prog Nucl Energ, 2007, 49: 37-60. DOI: 10.1016/j.pnucene.2006.08.003

[5] Hoisie A, Lubeck O and Wasserman H. Scalability analysis of multidimensional wavefront algorithms on large-scale SMP clusters. 7th Symposium on the Frontiers of Massively Parallel Computation, 1999, 4-15. DOI: 10.1109/FMPC.1999.750452

[6] Los Alamos National Laboratory. The ASCI Sweep3D Benchmark. http://wwwc3.lanl.gov/pal/software/sweep3d/

[7] Petrini F, Fossum G, Fernandez J, et al. Multicore surprises: lessons learned from optimizing Sweep3D on the cell broadband engine. IEEE International Parallel and Distributed Processing Symposium, 2007, 1-10. DOI: 10.1109/IPDPS.2007.370252

[8] Lubeck O, Lang M, Srinivasan R, et al. Implementation and performance modeling of deterministic particle transport (Sweep3D) on the IBM Cell/B.E.. Sci Program, 2009, 17: 199-208. DOI: 10.3233/SPR-2009-0266

[9] Gong C, Liu J, Gong Z, et al. Optimizing sweep3d for graphic processor unit. Lect Notes Comput Sc. 2010, 6081: 416-426. DOI: 10.1007/978-3-642-13119-6_{36}

[10] Gong C, Liu J, Chi L, et al. GPU accelerated simulations of 3D deterministic particle transport using discrete ordinates method. J Comput Phys, 2011, 230: 6010-6022. DOI: 10.1016/j.jcp.2011.04.010

[11] Davis K, Hoisie A, Johnson G, et al. A performance and scalability analysis of the BlueGene/L architecture. Proceedings of the ACM/IEEE SC2004 Conference on Supercomputing, IEEE Press, 2004, 41. DOI: 10.1109/SC.2004.8

[12] Barker K J, Davis K, Hoisie A, et al. Entering the petaflop era: The architecture and performance of Roadrunner. Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE Press, 2008, 1-11. DOI: 10.1109/SC.2008.5217926

[13] Wang Q, Xing Z, Liu J, et al. Parallel 3D deterministic particle transport on Intel MIC architecture. Proceedings of 2014 International Conference on High Performance Computing and Simulation (HPCS), IEEE Press, 2014, 186-192. DOI: 10.1109/HPCSim.2014.6903685

[14] Intel Corporation. Knights corner software developers guide revision 1.03, 2012, 11-35.

[15] Anthony S. Intel unveils 72-core x86 knights landing CPU for exascale supercomputing. http://www.extremetech.com/extreme/

[16] Wang Q, Liu J, Tang X, et al. Accelerating embarrassingly parallel algorithm on Intel MIC. Proceedings of 2014 International Conference on Progress in Informatics and Computing, IEEE Press, 2014, 213-218. DOI: 10.1109/PIC.2014.6972327

[17] Saule E and Catalyurek U V. An Early Evaluation of the Scalability of Graph Algorithms on the Intel MIC Architecture. In: Proceedings of 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, 2012, 18: 1629-1639. DOI: 10.1109/IPDPSW.2012.204

[18] Adams M L and Larsen E W. Fast iterative methods for discrete-ordinates particle transport calculations. Prog Nucl Energ, 2002, 40: 3-159. DOI: 10.1016/S0149-1970(01)00023-

[19] Williams S, Waterman A and Patterson D. Roofline: an insightful visual performance model for multicore architectures. Commun Acm, 2009, 52: 65-76. DOI: 10.1145/1498765.1498785

[20] McCalpin J D. STREAM: Sustainable memory bandwidth in high performance computers, 1995. http://www.cs.virginia.edu/stream/

Note: Figure translations are in progress. See original paper for figures.