# HYBRID MPI+OPENMP PARALLELIZATION OF IMAGE RECONSTRUCTION IN PROTON BEAM THERAPY ON MULTI-CORE AND MANY-CORE PROCESSORS

James Della-Giustina

School of Mathematics and Sciences
Community College of Baltimore County
800 S. Rolling Rd., Catonsville, MD, USA
jdella@umbc.edu

Carlos Barajas

Department of Mathematics and Statistics
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore, MD, USA
barajasc@umbc.edu

Matthias K. Gobbert

Department of Mathematics and Statistics
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore, MD, USA
gobbert@umbc.edu

Dennis S. Mackin

Department of Radiation Physics
The University of Texas MD Anderson Cancer Center
1515 Holcombe Blvd., Houston, TX, USA
dsmackin@mdanderson.org

Jerimy Polf

Department of Radiation Oncology
University of Maryland School of Medicine
655 W. Baltimore Street, Baltimore, MD, USA
jpolf@umm.edu

## ABSTRACT

The advantage of proton beam therapy is that the lethal dose of radiation is delivered by a sharp increase toward the end of the beam range, known as the Bragg peak (BP), with no dose delivered beyond. By using these characteristics of the BP, radiation dose to the tumor can be maximized, with greatly reduced radiation dose to the surrounding healthy tissue. If the secondary gamma rays that are emitted through interaction of the protons in the beam with atoms in the patient tissue could be imaged in (near) real-time during beam delivery, it could provide a means of visualizing the delivery of dose for verification of proper treatment delivery. However, such imaging requires very fast image reconstruction to be feasible. This project focuses on measuring the performance of a new parallel version of the CCI (Compton camera imaging) image reconstruction algorithm. We show two conclusions: (i) The new hybrid MPI+OpenMP parallelization of the code on the many-core Intel Xeon Phi KNL processor with 68 computational cores makes fast reconstruction times possible and thus enables the use of CCI in real time during treatment. (ii) A compute node with two of the latest multi-core Intel Skylake CPUs with 24 cores performs even better in a first comparison of both types of processors available on Stampede2 at the Texas Advanced Computing Center (TACC).

**Keywords:** Proton beam therapy, Image reconstruction, CCI algorithm, Intel Xeon Phi, Intel Skylake.

# 1 INTRODUCTION

Proton beam radiation treatment was first proposed by Robert Wilson in 1946 (Wilson 1946). While x-ray radiation delivers its lethal dose throughout the patient, proton beams reach their highest dose just before they stop, at what is called the Bragg peak (BP), with little or no dose delivered beyond this point. The depth at which the BP occurs in the patient can be controlled by carefully choosing the beams initial energy. This gives the physicians and radiotherapy specialists the ability to match the depth of the BP within the patient to that of the tumor. This allows for the accurate delivery of a lethal dose of radiation to the tumor while allowing the radiation exposure to the surrounding healthy tissues to be kept at acceptably low levels. While the potential to reduce irradiation of healthy tissue is a major advantage of proton beam radiotherapy, it is currently not possible to always utilize such an advantage. Over a full course of proton therapy (1 to 5 weeks), changes may occur inside the patient's body as both the surrounding tissue and tumor can swell or shrink in response to radiation, changing the size and relative position of the target area for treatment on a daily basis. Therefore, to target only tumor volume and avoid adjacent normal tissue (thus fully utilize the advantage of proton therapy), in vivo imaging of the treatment delivery would reduce uncertainties and allow the advantages of the Bragg peak to be fully exploited (Polf and Parodi 2015, Avila-Soto, Beri, Valenzuela, Wudenhe, Blenkhorn, Graf, Khuvis, Gobbert, and Polf 2015).

Nuclear scattering of the treatment beam protons produces secondary, gamma radiation. Because this gamma radiation is emitted only when the proton beam interacts with tissues, the origin of the gamma rays will map out the exact path of the proton beam through the body. The ability to image this gamma radiation would provide a method to image and verify the delivery of dose during each daily proton radiotherapy treatment. However, in order to provide these images to the physician and treatment specialists in real time during proton beam delivery, extremely fast image reconstruction techniques are needed. Thus, a limiting factor for implementing this type of imaging for treatment verification is the need for the image reconstruction code to produce results in essentially real-time, while the patient is undergoing treatment.

This paper investigates whether a new hybrid MPI+OpenMP parallelization of a statistical-reconstruction algorithm designed for secondary gamma imaging on modern multi-core or many-core processors can reliably reconstruct a sample image in a much less time than it takes to deliver a single beam (about 30 to 100 seconds) during daily treatment (Mackin, Peterson, Beddar, and Polf 2012). Previous work demonstrated that introducing parallelism with MPI, which allowed to use multiple compute nodes in a distributed-memory cluster, sped up the code to any desired time by using more nodes. But multi-node clusters cannot be housed in a treatment room and would require an off-site system with associated problems of reliable real-time access, such as risk of lost connections, in addition to additional training for physicians and imaging technicians, which make it simply impractical. Therefore, we investigate here if a single compute server, i.e., one node of a cluster, could accomplish the desired speedup. We show two conclusions: (i) We demonstrate that this is within reach and show that the many-core architecture of the Intel Xeon Phi KNL processor with 68 cores connected by a 2D mesh network can achieve this. Since the latest generation of the Phi, code-named Knights Landing (KNL), is a self-bootable CPU, it is realistically possible to use a computer with one KNL in a clinical setting. (ii) Moreover, we compare the performance of the KNL to a compute node with two of the latest multi-core Intel Skylake CPUs with 24 cores, which performs even better, provided MPI parallelism is used in the code. This is the first report of a comparison of both types of processors available on Stampede2 at the Texas Advanced Computing Center (TACC).

This paper is organized as follows: Section 2 contrasts available multi-core and many-core processors. Section 3 explains the reconstruction algorithm and its parallel implementation in hybrid MPI+OpenMP C/C++ code. Section 4 presents both application results of the image reconstruction and parallel performance studies on multi-core and many-core processors. Section 5 collects the conclusions of the work.

## 2    MULTI-CORE VS. MANY-CORE PROCESSORS

Modern CPUs feature growing number of computational cores, such as, for instance, 24 cores in an Intel 8160 Skylake CPU from 2017. Contrasted to these state-of-the-art multi-core CPUs with the most modern cores, many-core processors have even more cores, for instance, 68 cores in an Intel Xeon Phi 7250 Knights Landing (KNL) processor. The cores in a many-core processor typically feature a lower clock rate, e.g., 1.4 GHz vs. 2.1 GHz, but their larger number of cores with their rich connectivity in a 2D mesh network and the fact that significantly more high-performance memory is available on the chip itself offer the potential for better performance. Additionally, each core of a KNL is capable of running 4 hardware threads simultaneously. Investigating (i) this performance and (ii) comparing it to state-of-the-art multi-core CPUs are the purposes of this paper. Our results do not depend on the cluster Stampede2 at TACC, already since the focus is on single-node performance, but using both hardware in the same cluster ensures a fair comparison, for instance also with respect to software. We use the Intel compiler version 17.0.4 and the Intel MPI implementation version 17.0.4.

### 2.1  Many-Core Processor: Intel Xeon Phi 7250 KNL

Figure 1 illustrates the layout of one Intel Xeon Phi 7250 Knights Landing (KNL) processor, whose cores are clocked at 1.4 GHz. Each KNL core is capable of 4 hardware threads. This model of the KNL on Stampede2 contains 68 cores, which are arranged in pairs of two in a so-called tile that share the connection to the two-dimensional mesh network that connects all tiles. Each core has a L1 cache, while the two cores of a tile share an L2 cache. Each box without text inside (in red color) in Figure 1 indicates one tile with two cores and their shared L2 cache in the center of the tile between the cores. Inside the KNL are 16 GB of Multi-Channel DRAM (MCDRAM), arranged in 8 blocks of 2 GB each, shown on top and bottom of Figure 1. Outside of the socket, the KNL is connected to 96 GB of DDR4 memory, arranged in 6 DIMMs of 16 GB each, shown connected to their memory controllers (MC) on the left and right of Figure 1. This MCDRAM is located directly in the KNL and is up to five times faster than the on board DDR4 because of its three-dimensional arrangement (Sodani, Gramunt, Corbal, Kim, Vinod, Chinthamani, Hutsell, Agarwal, and Liu 2016).

The KNL uses three distinct memory modes, which determines how the processor treats the MCDRAM; as L3 cache, RAM, or a mixture of both. The KNL also offers three distinct cluster modes that ensure all cores possess the most current data among other processes, whether from main memory or MCDRAM. The differences between these cluster modes allow users to vary the amount of explicit control of memory management. For our studies, we chose to use the 'cache' memory mode and 'quadrant' cluster mode. We use the KNL in the Stampede2 cluster at the Texas Advanced Computing Center (TACC) at The University of Texas at Austin, which has over 4,200 nodes, each with one KNL processor.

### 2.2  Multi-Core Processor: Intel 8160 Skylake

The Stampede2 cluster at TACC has rolled out the new Intel Xeon Platinum 8160 Skylake CPUs in 2017, clocked at 2.1 GHz and with 32 MB L3 cache. These chips contain 24 cores, totaling 48 cores on a two-socket node. This node has 192 GB of memory available in 12 DIMMs of 16 GB DDR4 memory, surpassing the memory available on the KNL. However, its on-chip L3 cache does not compare to the 16 GB of high-speed MCDRAM that the KNL chip has. Stampede2 has 1,736 Skylake nodes, available since December 2017.
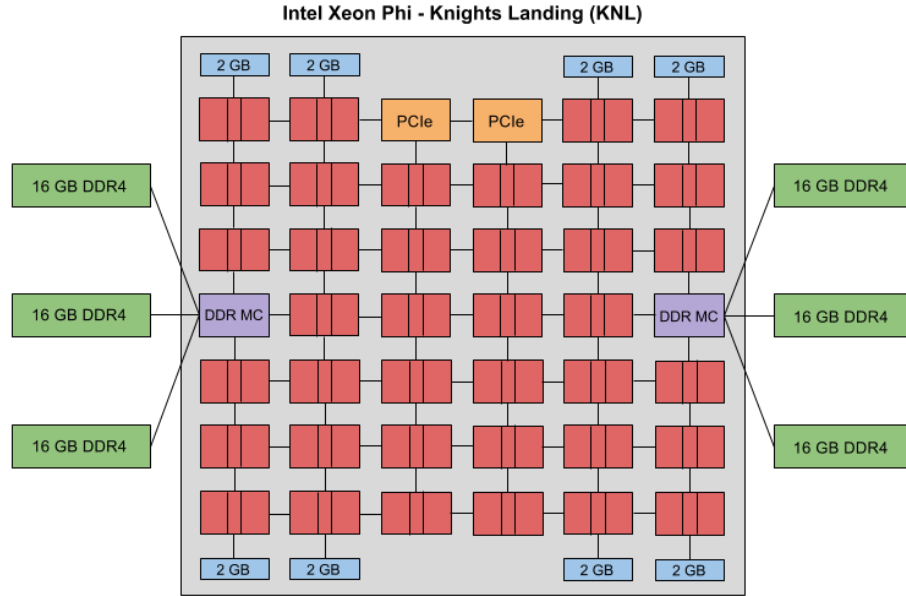
Figure 1: Intel Xeon Phi KNL schematic.

## 3 ALGORITHM AND IMPLEMENTATION

### 3.1 Compton Camera Imaging (CCI) Algorithm

During data collection, gamma rays scatter into a specially designed camera known as a "Compton Camera" (CC) which records the coordinates and energy deposited by each gamma ray that interacts with the CC. Each gamma ray must interact with the camera at least two times to be useful for imaging. The 3D coordinates and energies deposited by the gamma rays are stored in a data file which is used to initialize the conic image reconstruction software. A line is drawn between the two points of gamma ray interactions representing the central axis and using the energy deposition of the two interactions and the Compton scattering formula, an angle is calculated representing the half angle of the gammas interaction cone in Figure 2 (a). The cone's surface encompasses all the possible origins for that ray. After these cones have been constructed, a random point from the surface of the cone is chosen as an initial guess of the origin of the gamma ray. A 3D histogram, populated by the randomly chosen origins for each cone in Figure 2 (b), is used to estimate the gamma emission distribution for the reconstruction volume. Thus, one point on the surface represents each cone in the reconstruction.

The reconstruction algorithm, based on the Origin Ensembles method and later modifications (Andreyev, Sitek, and Celler 2011), uses a Markov process to iteratively move the representative points on the surface of the cones. After many iterations, the gamma emission density estimate formed by the representative points reaches a steady state and the algorithm is stopped.

### 3.2 Code Implementation

The algorithm is implemented in a C/C++ code with hybrid MPI+OpenMP parallelism. Structurally, parallel communication functions from MPI allow for the use of multiple nodes in a distributed-memory cluster, while OpenMP multi-threading parallelizes each MPI process within the shared-memory of a node. However, it is often the case, even on a single shared-memory node, that different combinations of MPI processes
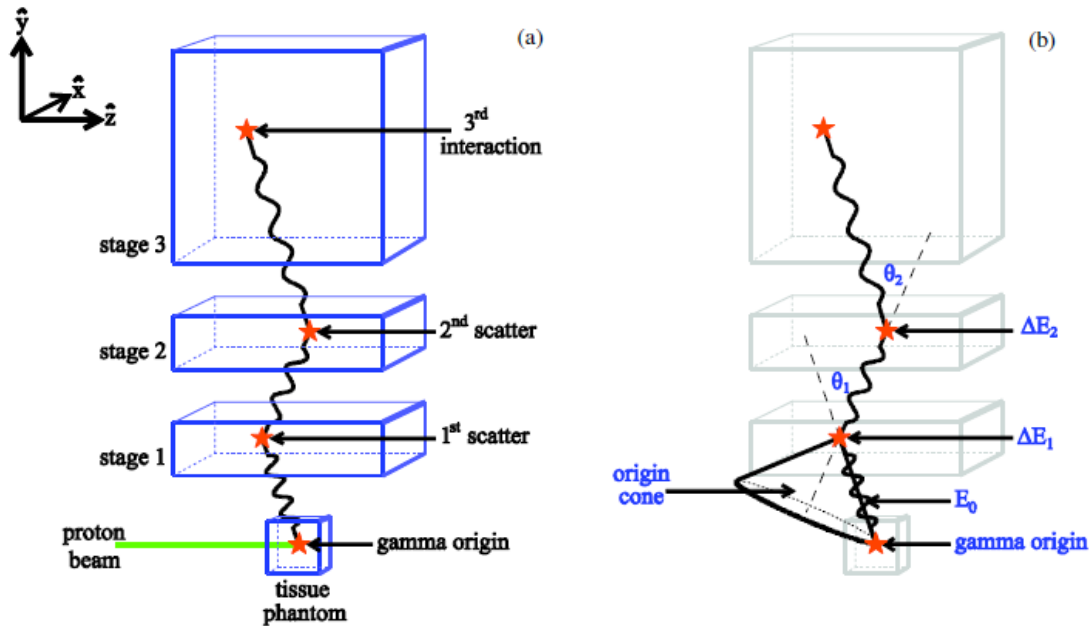
Figure 2: Gamma ray scatter and its image reconstruction.

and OpenMP threads per process result in different performance. In fact, for the KNL, it is a recommendation by TACC to use a modest number of MPI processes, combined with OpenMP multi-threading, instead of only OpenMP, even if the same number of hardware cores are used (Rosales, James, Gómez-Iglesias, Cazes, Huang, Liu, Liu, and Barth 2016). Thus, it is vital to have hybrid MPI+OpenMP code available for best performance.

First, a configuration file is parsed and data loaded from an input file with physical measurements. Important parameters include the total number of cones used, histogram coordinate boundaries in the $x$, $y$, and $z$ directions, the total number of bins in the $x$, $y$, $z$ directions, and the total number of iterations; notice that the number of iterations is fixed here by trial and error based on observing that the fraction of origins moved during each iteration stops changing. The total number of density histograms used for estimation is also set in the configuration file.

Before iterating, the collection of cones are distributed to the MPI processes as equally as possible. A global collection of before-iteration conic likely origins is generated and used for the initial population of the local density histograms on each process. An empty collection of after-iteration likely origins is created as well but not populated.

The CCI algorithm iteration is nearly identical to the original OpenMP algorithm as described in (Mackin, Peterson, Beddar, and Polf 2012). On each MPI process, OpenMP threads are started up using a `parallel for` pragma to iterate over the process' cones. During the iteration two situations will occur, either the thread will determine a cone's likely origin should be moved, or the cone's likely origin should remain where it was. If the cone's likely origin is to be moved, the new location is recorded into the process's local copy of the after-iteration collection. If the cone's likely origin is to remain the same, then the original likely origin is recorded into the process's local copy of the after-iteration collection.

At the end of the iteration, the local copies of the after-iteration collection are merged into a global version. Then all likely origins are updated for the local density histograms on each process. After all iterations have completed, all conic data is sent to the MPI process 0 for output to files.

## 4 RESULTS

### 4.1 Iterative Image Reconstruction

The CCI algorithm uses iterations to progressively compute the most likely origins of the 100,000 cones that were measured by the CC during the proton beam irradiation of a water phantom. The quality of the CCI algorithm is controlled also by the choice of the number of shifted histograms used, for which $h_{total} = 200$ has proven a good choice in the past. As more iterations are performed the iterated images become more clearly defined, as seen in Figure 3. Although at 100 iterations, the image has some shape, as the number of iterations increases, this shape becomes more accurate. Additionally as the number of iterations increases, the granularity of the results improve until the stopping point of 600 iterations, where a distinct beam can be seen.
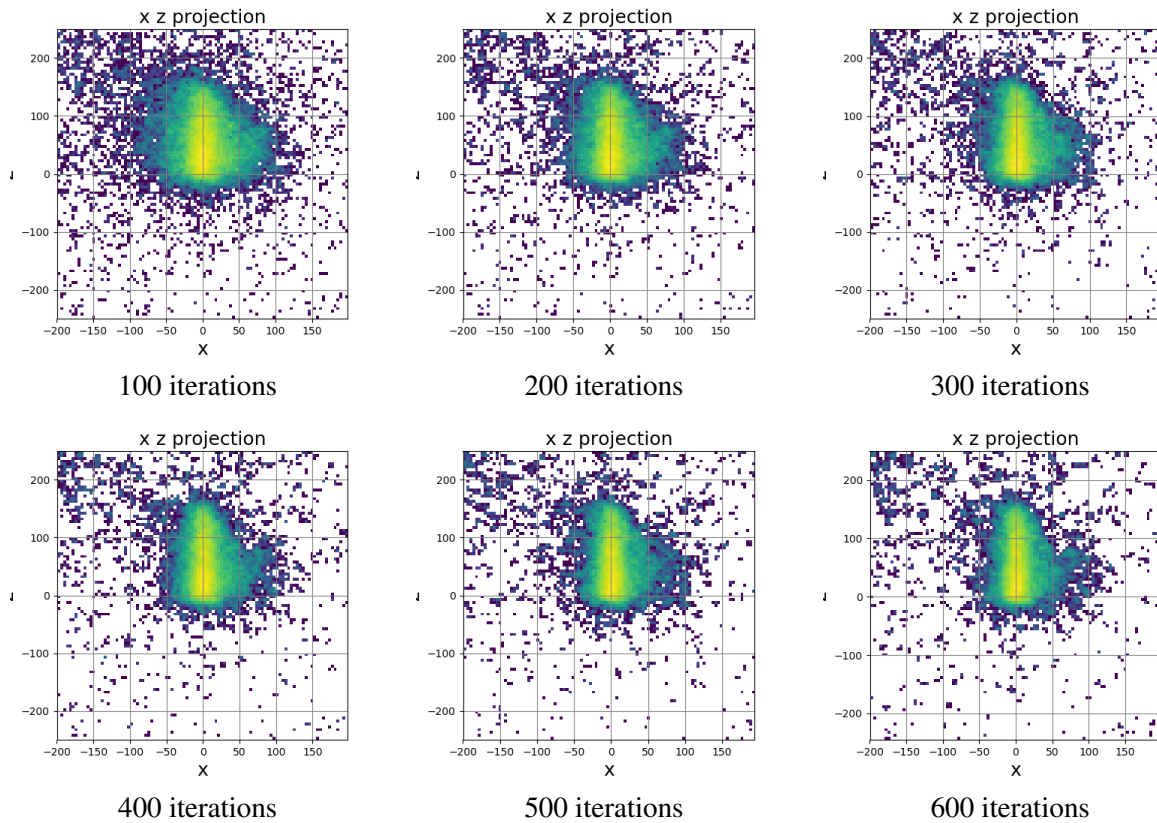


Figure 3: Reconstructed images ($x$-$z$-projections) at 100, ..., 600 iterations of the CCI algorithm using $h_{total} = 200$ shifted histograms, accessed by $p = 1$ process and $h_p = 200$ histograms per process.

The total number of shifted histograms, also referred to as total histograms controlled in the configuration file and denoted as $h_{total}$, is split among the $p$ MPI processes, so that each process has the same number of shifted histograms per process denoted as $h_p$. The quality of the image reconstruction is controlled by the total number of $h_{total} = h_p\, p$ histograms; since the algorithm is iterative and involves random number sequences, and is thus not completely deterministic, it is possible to compare also results, for which $h_{total}$ agrees only approximately. This is tested in Figure 4, which shows final images at 600 iterations of the CCI algorithm with the same number of $h_{total} = h_p\, p = 200$ histograms as in Figure 3, but accessed by different possible combinations of $p$ and $h_p$. Comparing the 600 iteration image in Figure 3 with $p = 1$ process to the images in Figure 4 with $p = 2, 4, 8, 16, 32, 64$ processes, we can observe that any combination of $p$ and $h_p$ that satisfies approximately $h_{total} = h_p\, p \approx 200$ gives the same quality of the result.
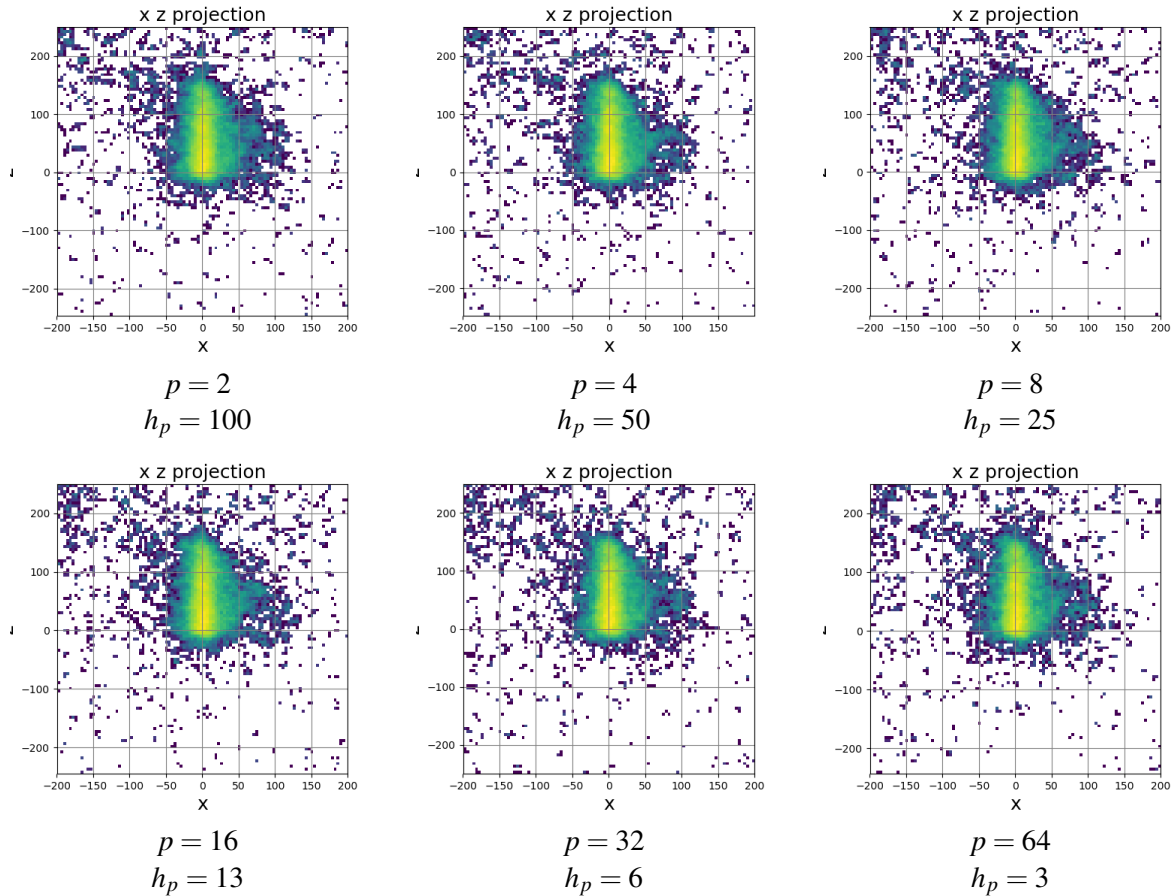


Figure 4: Reconstructed images (*x-z*-projections) at 600 iterations of the CCI algorithm using $h_{total} = 200$ shifted histograms, accessed by various combinations of $p$ processes and $h_p$ histograms per process.

## 4.2 Single-Node Performance

In the clinical setting of a treatment room, the only realistic equipment is a computer server that is equivalent to a single, possibly high-end, node in a computer cluster, such as the ones used in our studies.

Tables 1 and 2 compare the timing performance of the studies in Section 4.1 with 100,000 cones and 600 iterations of the CCI algorithm. The times reported measure the performance of the variable portion of the algorithm, that is, the iterations of the CCI algorithm. There are additional times for initialization and for output (for post-processing), which are independent of the parallelization; however, they turn out to be significantly different in scale for the two platforms, namely about 40 seconds for the KNL and about 10 seconds for the Skylake.

Table 1 shows the performance results on one KNL processor. Recall that each KNL on Stampede2 has 68 computational cores. Thus, to take advantage of all cores when using $p$ MPI processes, hybrid MPI+OpenMP code uses the number of threads per MPI process $t_p$ chosen such that the total number of threads $t_{total} = t_p p = 68$. To be precise, the above holds exactly true for $p = 1, 2, 4$ only. For $p = 8, 16, 32, 64$, for which the product $t_{total} = t_p p = 68$ cannot hold exactly, we actually idle some cores by using $t_{total} = t_p p = 64$ for these cases. Each core of a KNL is capable of running simultaneously up to 4 hardware threads. Thus, we can also choose the threads per MPI process $t_p$ such that the total number of threads $t_{total} = t_p p = 136$ or 272 for running 2 or 4, respectively, threads per KNL core. In this way, each row of Table 1 uses the number of total threads $t_{total}$ indicated in the first column. The difference between the columns lies in how the same number of $t_{total}$ threads is accessed by combining $p$ MPI processes with $t_p$ threads per process. The total number of threads are as scattered as possible across all 68 hardware cores of the 34 tiles.

Each row of results in Table 1 indicates that using larger numbers of MPI processes $p$ yields better timings, except some outlying behavior at $p = 32$ for the cases of 2 and 4 threads per MPI process. The improving run times, certainly for the case of 1 thread per MPI process, indicates that the 2D mesh network can facilitate communications between MPI processes very efficiently. The improvement of run times with growing MPI process count is not just explained by potential advantages in behavior of shared-memory multi-threading, but also by the algorithmic difference of correspondingly different numbers of shifted histograms per process. Recall these seven cases are the same simulations as in Figure 3 with $p = 1$ and in Figure 4 with $p = 2, 4, 8, 16, 32, 64$, and we established already that their simulations yield equivalent, acceptable results.

The purpose of Table 1 is to determine if there is any advantage to running multiple hardware threads per KNL core. As it becomes clear, our experience is consistent with many other recommendations including TACC documentation (Rosales, James, Gómez-Iglesias, Cazes, Huang, Liu, Liu, and Barth 2016), that it is typically most beneficial to only run 1 or 2 hardware threads per KNL core. In fact, for this code, running only 1 hardware thread per KNL core is the best choice in nearly all cases. This establishes that the optimal results for this test case of the CCI algorithm is the first result row of results in Table 1; we will use these results to compare to performance on other hardware.

Table 2 shows the direct comparison of run times of the best results of the KNL simulations in Table 1 with corresponding studies of a node with two multi-core Intel Skylake CPUs. Specifically, the KNL results are those running 1 hardware thread per KNL core from the first result row of Table 1. One node with two 24-core Intel Skylake CPUs on Stampede2 allows for a maximum of 48 threads. The second result row in Table 2 shows the timings using $t_{total} = 32$ total threads, with $t_p$ chosen such that $t_{total} = t_p p = 32$; we stay with the power of 2 for consistency in this row, although 48 cores are available. The third result row in Table 2 shows the timings using all available $t_{total} = 48$ total threads, with $t_p$ chosen such that $t_{total} = t_p p = 48$; in the $p = 32$ column for this row, we actually record the result for $p = 48$ and $t_p = 1$. The "N/A" indicate that the number of cores is not available for the hardware. Utilizing all 48 cores, that is,

Table 1: Observed wall clock time in seconds for reconstruction at 600 iterations of the CCI algorithm using $h_{total} = 200$ shifted histograms, accessed by various combinations of $p$ processes and $h_p$ histograms per process. The number $t_p$ of OpenMP threads per MPI process is chosen such that the total number of threads is the specified $t_{total}$.

| MPI processes $p =$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|
| Histograms per MPI process $h_p =$ | 200 | 100 | 50 | 25 | 13 | 6 | 3 |
| KNL with 1 hardware thread per core: $t_{total} = 68$ | 156 | 78 | 39 | 21 | 13 | 9 | 8 |
| KNL with 2 hardware threads per core: $t_{total} = 136$ | 173 | 84 | 41 | 34 | 20 | 67 | 8 |
| KNL with 4 hardware threads per core: $t_{total} = 272$ | 181 | 103 | 57 | 31 | 19 | 134 | 11 |

$t_{total} = 48$, produces faster results for smaller numbers of MPI processes $p$, but as $p$ increases, performance improvements are negligible. Notice that on both the KNL and Skylake, as MPI processes $p$ increase, so does performance. In fact, on the Skylake node, performance profits most from larger numbers of MPI processes $p$ to the point of OpenMP single-threading ($t_p = 1$ per MPI process). Notice again that this is not just the effect of parallel performance, but also due to the algorithmic design of distributing the $h_{total}$ shifted histograms to the $p$ MPI processes. While the KNL's best performance is able to complete iterating in 8 seconds, the Skylake is able to achieve this in 3 seconds. In every case, the Skylake significantly outperforms the KNL no matter the combination of $p$ and $t_p$ are chosen.

Table 2: Observed wall clock time in seconds for reconstruction at 600 iterations of the CCI algorithm using $h_{total} = 200$ shifted histograms, accessed by various combinations of $p$ processes and $h_p$ histograms per process. The number $t_p$ of OpenMP threads per MPI process is chosen such that the total number of threads is the specified $t_{total}$, which is the optimal value for each hardware platform.

| MPI processes $p =$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|
| Histograms per process $h_p =$ | 200 | 100 | 50 | 25 | 13 | 6 | 3 |
| KNL with 1 hardware thread per core: $t_{total} = 68$ | 156 | 78 | 39 | 21 | 13 | 9 | 8 |
| Two 24-core Intel Skylake: $t_{total} = 32$ | 62 | 19 | 11 | 7 | 5 | 3 | N/A |
| Two 24-core Intel Skylake: $t_{total} = 48$ | 51 | 16 | 9 | 5 | 4 | 3 | N/A |

## 5 CONCLUSIONS

First, the application results confirm that the parallelized algorithm takes advantage of the problem structure correctly in the way that it distributes the total number of shifted histograms to the parallel processes. Second, for these simulations with equivalent application results, we show the power of the new hybrid MPI+OpenMP parallelization of the code to potentially dramatically reduce run times on both the Intel Xeon Phi KNL and new Intel Skylake CPUs. In the present results, it turned out that this code on the Skylake hardware did not profit from multi-threading, but this feature will likely be necessary if using multiple nodes of a distributed-memory cluster. Third, the performance results with the new Intel Skylake CPUs are sufficiently fast to move the image reconstruction close to the timeframe suitable for the clinical environment of a patient having to lie still for this amount of time, even with some additional time for setup, output, and post-processing. This result is far more achievable with the Skylake, it seems, as the additional time for setup and post-processing of 40 seconds on the KNL is significantly larger than the 10 seconds on the Skylakes.

In summary, while we see that the many-core KNL allows for impressive scaling, a dual-socket node with the most modern Intel Skylake CPUs yet beats the KNL, provided MPI parallelism is used in the code. This is both due to the core count being nearly as high as on the KNL and the higher clock rate of each core. The KNL's advantages should be the ability to run several threads per KNL core and the faster on-chip

memory (as opposed to Skylake accessing main memory on the node), but they are not sufficient to beat the Skylakes. There also seems to be less efficient connection to the node's main memory, given the much larger additional time for setup and post-processing on the KNL. TACC has just released the Intel Skylake CPUs with 24 cores in December 2017, and we hope this paper to be the first to contrast the performance of both processor types, many-core KNL and multi-core Skylake, available on Stampede2 for the near future.

## ACKNOWLEDGMENTS

## REFERENCES

Andreyev, A., A. Sitek, and A. Celler. 2011. "Fast image reconstruction for Compton camera using stochastic origin ensemble approach". *Med. Phys.* vol. 38 (1), pp. 429–438.

Avila-Soto, F. X., A. N. Beri, E. Valenzuela, A. Wudenhe, A. R. Blenkhorn, J. S. Graf, S. Khuvis, M. K. Gobbert, and J. Polf. 2015. "Parallelization for Fast Image Reconstruction using the Stochastic Origin Ensemble Method for Proton Beam Therapy". Technical Report HPCF–2015–27, UMBC High Performance Computing Facility, University of Maryland, Baltimore County.

Mackin, D., S. Peterson, S. Beddar, and J. Polf. 2012. "Evaluation of a stochastic reconstruction algorithm for use in Compton camera imaging and beam range verification from secondary gamma emission during proton therapy". *Phys. Med. Biol.* vol. 57 (11), pp. 3537–3553.

Polf, J. C., and K. Parodi. 2015. "Imaging particle beams for cancer treatment". *Physics Today* vol. 68 (10), pp. 28–33.

Rosales, C., D. James, A. Gómez-Iglesias, J. Cazes, L. Huang, H. Liu, S. Liu, and W. Barth. 2016. "KNL Utilization Guidelines". Technical Report TR–16–03, Texas Advanced Computing Center, The University of Texas at Austin.

Sodani, A., R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu. 2016. "Knights Landing: Second-Generation Intel Xeon Phi Product". *IEEE Micro* vol. 32 (2), pp. 34–46.

Towns, J., T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr. 2014. "XSEDE: Accelerating Scientific Discovery". *Comput. Sci. Eng.* vol. 16 (5), pp. 62–74.

Wilson, R. R. 1946. "Radiological Use of Fast Protons". *Radiology* vol. 47 (5), pp. 487–491.

## AUTHOR BIOGRAPHIES

**JAMES DELLA-GIUSTINA** is an undergraduate student pursuing a B.S. in Computer Science, currently at the Community College of Baltimore County. His interests lie in parallel computing, embedded systems, and cryptography. His email address is jdella@umbc.edu.

**CARLOS BARAJAS** is a graduate student pursuing his Ph.D. in Applied Mathematics at the University of Maryland, Baltimore County. He received his B.A. in Computer Science and Pure Mathematics from Olivet College. His email address is barajasc@umbc.edu.

**MATTHIAS K. GOBBERT** received his Ph.D. from Arizona State University. He is a Professor in the Department of Mathematics and Statistics at the University of Maryland, Baltimore County and currently serves as program director in the Division of Mathematical Sciences at the National Science Foundation. His email address is gobbert@umbc.edu.

**DENNIS S. MACKIN** is currently Assistant Professor at the University of Texas, MD Anderson Cancer Center. He holds a Ph.D. in High Energy Particle Physics from Rice University. His email address is dsmackin@mdanderson.org.

**JERIMY POLF** is Associate Professor at the Marlene and Stewart Greenebaum Cancer Center at the University of Maryland, Baltimore. He holds a Ph.D. in Photonics from Oklahoma State University and is board certified in Medical Physics by the American Board of Radiology. His email address is jpolf@umm.edu.