Benchmarking Parallel Performance on Many-Core Processors

Bryant C. Lam, Ajay Barboza, Ravi Agrawal, Alan D. George, and Herman Lam

NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611-6200
{blam, barboza, ragrawal, george, hlam}@chrec.org

Abstract. With the emergence of many-core processor architectures onto the HPC scene, concerns arise regarding the performance and productivity of numerous existing parallel-programming tools, models, and languages. As these devices begin augmenting conventional distributed cluster systems in an evolving age of heterogeneous supercomputing, proper evaluation and profiling of many-core processors must occur in order to understand their performance and architectural strengths with existing parallel-programming environments and HPC applications. This paper presents and evaluates the comparative performance between two many-core processors, the Tilera TILE-Gx8036 and the Intel Xeon Phi 5110P, in the context of their applications performance with the SHMEM and OpenMP parallel-programming environments. Several applications written or provided in SHMEM and OpenMP are evaluated in order to analyze the scalability of existing tools and libraries on these many-core platforms. Our results show that SHMEM and OpenMP parallel applications scale well on the TILE-Gx and Xeon Phi, but heavily depend on optimized libraries and instrumentation.

Keywords: PGAS, SHMEM, OpenMP, many-core, parallel programming, performance analysis, high-performance computing, parallel architectures

1 Introduction

With the emergence of many-core processors into the high-performance computing (HPC) scene, there is strong interest in evaluating and evolving existing parallel-programming models, tools, and libraries. This evolution is necessary to best exploit the increasing single-device parallelism from multi- and many-core processors, especially in a field focused on massively distributed supercomputers. Although many-core devices offer exciting new opportunities for application acceleration, these devices need to be properly evaluated between each other and the conventional servers they potentially supplement or replace.

A very popular parallelization method for HPC applications is a hybrid mix of shared-memory threads with OpenMP [3] for intra-node communication between processor cores and single-program, multiple-data (SPMD) processes with MPI [5] for inter-node communication. Partitioned global address space (PGAS) languages and libraries, however, are experiencing rising interest with their potential to provide high-performance around a straightforward memory and communication model. Notable members of the PGAS family include Unified Parallel C (UPC), X10, Chapel, Co-Array Fortran (CAF), Titanium, and SHMEM [9].

In this paper, we present and evaluate the performance of SHMEM and OpenMP applications on two current-generation many-core devices, the Tilera TILE-Gx and the Intel Xeon Phi. The SHMEM applications are evaluated with two library implementations: the OpenSHMEM reference implementation [11] on both platforms, and TSHMEM [7]—an OpenSHMEM library optimized specifically for Tilera many-core processors—on the TILE-Gx. OpenMP implementations are provided by the native compiler for each platforms. Results from these applications emphasize comparative performance of our many-core devices and the effectiveness of each parallel-programming environment.

The remainder of the paper is organized as follows. Section 2 provides background on the SHMEM library and OpenSHMEM efforts, our previous research with TSHMEM (SHMEM for Tilera processors), OpenMP, and brief architectural descriptions of the Tilera TILE-Gx and Intel Xeon Phi. Section 3 presents and evaluates several SHMEM and OpenMP applications on the two many-core platforms. Finally, Section 4 provides conclusions and directions for future work.

2 Background

This section provides brief background of the parallel-programming environments under analysis (SHMEM, OpenSHMEM, TSHMEM, and OpenMP) and the Tilera and Intel many-core platforms that will execute these applications.

2.1 SHMEM and OpenSHMEM

The SHMEM communication library adheres to a strict PGAS model whereby each cooperating parallel process (also known as a processing element, or PE) consists of a shared symmetric partition within the global address space. Each symmetric partition consists of symmetric objects (variables or arrays) of the same size, type, and relative address on all PEs. Originally developed to provide shared-memory semantics on the distributed-memory Cray T3D supercomputer, SHMEM closely models SPMD via its symmetric, partitioned, global address space.

There are two types of symmetric objects that can reside in the symmetric partitions: static and dynamic. Static variables reside in the heap segment of the program executable and are allocated during link time. These static variables, when parallelized as multiple processes, appear at the same virtual address to all processes running the same executable, thus ensuring its symmetry across all partitions. Dynamic symmetric variables, in contrast, are allocated at runtime on all PEs via SHMEM's dynamic memory allocation function *shmalloc()*. These dynamic variables, however, may or may not be allocated at the same virtual address on all PEs, but are typically at the same offset relative to the start of each symmetric partition.

SHMEM provides several routines for explicit communication between PEs, including one-sided data transfers (puts and gets), blocking barrier synchronization, and collective operations. In addition to being a high-performance, lightweight library, SHMEM has historically provided for atomic memory operations not available in popular library alternatives until recently (e.g., MPI 3.0).

Due to the lightweight nature of SHMEM, commercial variants have emerged from vendors such as Cray, SGI, and Quadrics. Application portability between variants, however, proved difficult due to different functional semantics, incompatible APIs, or system-specific implementations. This situation had regrettably fragmented developer adoption in the HPC community. Fortunately, SHMEM has seen renewed interest in the form of OpenSHMEM, a community-led effort to create a standard specification for SHMEM functions and semantics. Version 1.0 of the OpenSHMEM specification has already seen commercial adoption by vendors such as Mellanox [8].

2.2 GASNet and the OpenSHMEM Reference Implementation

The OpenSHMEM community provides a reference implementation of their library with primary source-code contributions from the University of Houston and Oak Ridge National Laboratory [11]. This reference implementation is compliant with version 1.0 of the OpenSHMEM specification and is implemented atop GASNet [2], a low-level networking layer and communication system for supporting SPMD parallel-programming models such as PGAS. GASNet defines a core and an extended networking API that are implemented via conduits. These conduits enable support for numerous networking technologies and systems. By leveraging GASNet's conduit abstraction, the OpenSHMEM reference implementation is portable to numerous cluster-based systems.

2.3 TSHMEM for Tilera Many-Core Processors

Our prior work with OpenSHMEM involved the design and evaluation of a library called TSHMEM (TileSHMEM) for Tilera many-core processors [7]. Along with improving developer productivity via the lightweight SHMEM API, TSHMEM delivers a high-performance many-core programming library and enables SHMEM application portability for the Tilera TILE-Gx and TILE*Pro* architectures. With the purpose of leveraging many-core capabilities and optimizations, TSHMEM is built atop Tilera-provided libraries and evaluated via microbenchmarking in order to demonstrate realizable performance and minimal overhead between the underlying libraries and TSHMEM functionality. Notable optimizations in TSHMEM include leveraging the Tilera on-chip mesh network for very-low-latency barriers. Optimization and evolution of TSHMEM continues as we research and experiment with functionality for current and future OpenSHMEM standards.

2.4 OpenMP

The OpenMP specification defines a collection of library routines, compiler directives, and environment variables that enable application parallelization via multiple threads of execution [3]. Standardized in 1997, OpenMP has been widely adopted and is portable across multiple platforms.

OpenMP commonly exploits SMP architectures by enabling both data-level and thread-level parallelism. Parallelization is typically achieved via a fork-and-join approach controlled by compiler directives whereby a master thread will fork several child threads when encountering an OpenMP parallelization section. The child threads may be assigned to different processing cores and operate independently, thereby sharing the computational load with the master. Threads are also capable of accessing shared-memory variables and data structures to assist computation. At the end of each parallel section, child threads are joined with the master thread and the parallel section closes. The master thread continues on with sequential code execution until another parallel section is encountered.

While other multi-threading APIs exist (e.g., POSIX threads), OpenMP is comparatively easier to use for developers that desire an incremental path to application parallelization for their existing sequential code. With the emergence of many-core processors such as the TILE-Gx and Xeon Phi, OpenMP is evolving to become a viable choice for single-device supercomputing tasks.

2.5 Tilera TILE-Gx

Tilera Corporation, based in San Jose, California, develops commercial many-core processors with emphases on high performance and low power in the cloud computing, general server, and embedded devices markets. Each Tilera many-core processor is designed as a scalable 2D mesh of tiles, with each tile consisting of a processing core and cache system. These tiles are attached to several on-chip networks via non-blocking cut-through switches. Referred to as the Tilera iMesh (intelligent Mesh), this scalable 2D mesh consists of dynamic networks that provide data routing between memory controllers, caches, and external I/O and enables developers to explicitly transfer data between tiles via a low-level user-accessible dynamic network.

Our research focuses on the current-generation Tilera TILE-Gx8036. The TILE-Gx is Tilera's new generation of 64-bit many-core processors. Differentiated by a substantially redesigned architecture from its 32-bit predecessor—the TILE*Pro*—the TILE-Gx exhibits upgraded processing cores, improved iMesh interconnects, and novel on-chip accelerators. Each 64-bit processing core is attached to five dynamic networks on the iMesh. The TILE-Gx8036 (a specific model of the TILE-Gx36) has 36 tiles, each consisting of a 64-bit VLIW processor with 32k L1i, 32k L1d, and 256k L2 cache. Furthermore, the L2 cache of each core is aggregated to form a large unified L3 cache. The TILE-Gx8036 offers up to 500 Gbps of memory bandwidth and over 60 Tbps of on-chip mesh interconnect bandwidth. An operating frequency from 1.0 to 1.2 GHz allows this processor to perform up to 750 billion operations per second at 10 to 55W (22W typical)

TDP. Other members of the TILE-Gx family include the TILE-Gx9, TILE-Gx16, and TILE-Gx72. In addition, the TILE-Gx includes hardware accelerators not found on previous Tilera processors: mPIPE (multicore Programmable Intelligent Packet Engine) for wire-speed packet classification, distribution, and load balancing; and MiCA (Multicore iMesh Coprocessing Accelerator) for cryptographic and compression acceleration.

2.6 Intel Xeon Phi

The Xeon Phi is Intel's line of many-core coprocessors. With processing cores based on the original Intel Pentium architecture, the Xeon Phi architecture is comprised of up to 61 x86-like cores with an in-order memory model on a ring-bus topology. Its performance strength is derived from the 512-bit SIMD vector units on each core, providing maximum performance to highly vectorized applications. With four hardware-thread units per core (up to 244), the Xeon Phi can theoretically achieve more than 1 TFLOPS of double-precision performance.

Each core of the Xeon Phi consists of an x86-like processor with hardware support for four threads and 32k L1i, 32k L1d, and 512k L2 caches. The unification of the L2 caches are globally available to the cores as an L3 cache and are kept coherent via a globally distributed tag directory. Several wide high-bandwidth bidirectional ring interconnects connect the cores to each other and to the on-board GDDR5 memory modules. Data movement is facilitated by this bidirectional hardware ring bus. The main BL ring carries 64-byte wide data in each direction while the narrower AD and AK carry address and coherence information respectively.

With the Xeon Phi, Intel aims to offer a general-purpose application co-processor and accelerator for large-scale heterogeneous systems. Housed in a GPU form factor, the Xeon Phi attaches to the PCIe bus of a standard host server and provides application acceleration via three modes of operation: offload for highly parallel work while the host processor executes typically serial work, symmetric for parallel work sharing between the host and coprocessor, and native for only-coprocessor application executions. Multiple Xeon Phi coprocessors can be attached to a single server for very high computational throughput per unit area. By supporting tools and libraries (e.g., OpenMP, Intel Clik, Intel MKL) available for code acceleration on Xeon processors, code portability to the Xeon Phi is relatively straightforward. Further performance optimizations for the Xeon Phi generally enable higher parallelism for the application on other platforms.

Our research focuses on the Intel Xeon Phi 5110P coprocessor. This coprocessor model is comprised of 60 cores with 240 hardware threads, 30 MB of on-chip cache, and 8 GB GDDR5 memory with peak bandwidth of 320 GB/s. Operating at 1.053 GHz, this passively-cooled coprocessor operates at 225W TDP.

3 SHMEM and OpenMP Performance Studies

SHMEM and OpenMP are highly amenable programming environments for SMP architectures due to their shared-memory semantics. With many-core processors emerging onto the HPC scene, developers are exceedingly interested in the performance and scalability of their applications for these devices. This section attempts to fairly analyze several applications, written in both SHMEM and OpenMP, on the TILE-Gx and Xeon Phi many-core processors. We then analyze several SHMEM-only applications to showcase performance differences between SHMEM-library implementations and conclude the section with observational experiences with both SHMEM and OpenMP, emphasizing their respective strengths and weaknesses.

3.1 Experimental Setup

The many-core platforms targeted by our research are the Tilera TILEmpower-Gx stand-alone server with a single TILE-Gx8036 [10] operating at 1.0 GHz, and the Intel Xeon Phi 5110P [6] passively-cooled PCIe card operating at 1.053 GHz attached to a standard host server. While the Xeon Phi is also designed as a coprocessor capable of offloaded workloads when paired with a Xeon host processor, we conduct our application executions with the Xeon Phi in native-execution mode only. All workloads execute only on the Xeon Phi with no significant influence from the host processor.

The SHMEM implementations under analysis include the OpenSHMEM reference implementation version 1.0d (referred to afterward as simply OpenSHMEM) and our TSHMEM library. The underlying API in OpenSHMEM is provided by GASNet version 1.20.2. GASNet was straightforwardly cross-compiled for the TILE-Gx, but cross-compiling for the Xeon Phi required minor source-code modifications to resolve missing x86 assembly instructions from the instruction-set architecture (ISA). Because the Xeon Phi has an in-order memory model, several x86 instructions related to memory fencing (i.e., sfence, lfence, mfence) are not necessary and are not included in the Xeon Phi ISA. These memory-fence instructions in GASNet were replaced with compiler barriers that effectively resolve into no-ops. In contrast, the OpenSHMEM library cleanly cross-compiles for both the TILE-Gx and Xeon Phi due to its use of the GASNet API. We leverage the GASNet SMP conduit for our experiments with OpenSHMEM. The second SHMEM implementation is our TSHMEM library which natively builds for the TILE-Gx. Due to significant use of Tilera libraries, TSHMEM portability to other platforms is under investigation for future work.

OpenMP is supported on both platforms via their native compiler. The TILE-Gx uses GCC version 4.4.6. OpenMP programs are compiled for the Xeon Phi with ICC version 13.0.1. Unless otherwise mentioned, all SHMEM and OpenMP applications are compiled with O2 optimizations.

3.2 SHMEM and OpenMP Applications

Our performance analysis of parallel-programming environments consists of three applications written each in both SHMEM and OpenMP. These applications are presented as follows: matrix multiply, linear curve fitting, and exponential curve fitting. In conducting our performance evaluation, we attempt to be impartial by evaluating how well an application's algorithm will map to the programming environment. When possible, specific optimizations were made only when the computational algorithm remained unchanged for both versions of the application. Scalability results are presented with increasing number of PEs, where PEs are either processes in SHMEM or threads in OpenMP. Serial-baseline executions were written only in the C programming language and are used for both OpenMP and SHMEM scalability at 1 PE. Results are reported in execution times for each application with power-of-two PEs as well as the realistic maximum number of PEs per device: 36 for the TILE-Gx and 240 for the Xeon Phi (60 cores).

Matrix Multiply. Matrix multiplication is a fundamental kernel in HPC and the computational sciences. The matrix-multiplication algorithm chosen for instrumentation was a partial-row dissemination with loop-interchange optimization for three matrices: $C = A \times B$. Each PE is partitioned a block of sequential rows to compute the partial result. In the case of OpenMP, the A, B, and C matrices are shared among the threads via compiler directives. Because of SHMEM's symmetric heap, the A and C matrices can be easily partitioned among the PEs, but each PE receives a private copy of the B matrix due to the pattern of computation. There are other parallelization strategies that do not require private matrix copies, but the pattern of computation and communication would have differentiated from the OpenMP version. In addition to row dissemination, loop interchange can easily occur since each matrix element in C has no data dependency with its other elements. By interchanging the inner-most loop with one of its outer loops, locality of reference and cache-hit rates drastically increase.

Execution times for SHMEM and OpenMP matrix multiplication are presented in Figure 1a. While the TILE-Gx execution times are significantly longer than the Xeon Phi's, normalizing the results with device power consumption will show more competitive conclusions. This power normalization, however, is only mentioned casually and will be investigated in the future.

For both platforms, OpenMP, OpenSHMEM, and TSHMEM execution times scale with each other up to 8 PEs. At this point, OpenSHMEM begins to stop scaling as closely with OpenMP and TSHMEM and eventually increases in execution time with higher PE counts. The performance of TSHMEM, however, follows closely with OpenMP for the TILE-Gx. Due to the Xeon Phi's 4-way SMT processing cores, each core's L1 and L2 caches are shared amongst its hardware-thread engines. Our Xeon Phi is equipped with 60 cores, therefore scalability beyond 60 is highly impacted by cache-hit rates as threads begin competing for per-core resources. For this specific OpenMP matrix multiplication, the Xeon Phi stops scaling around 128 threads. For the TILE-Gx, TSHMEM outperforms OpenSHMEM for all PE counts.

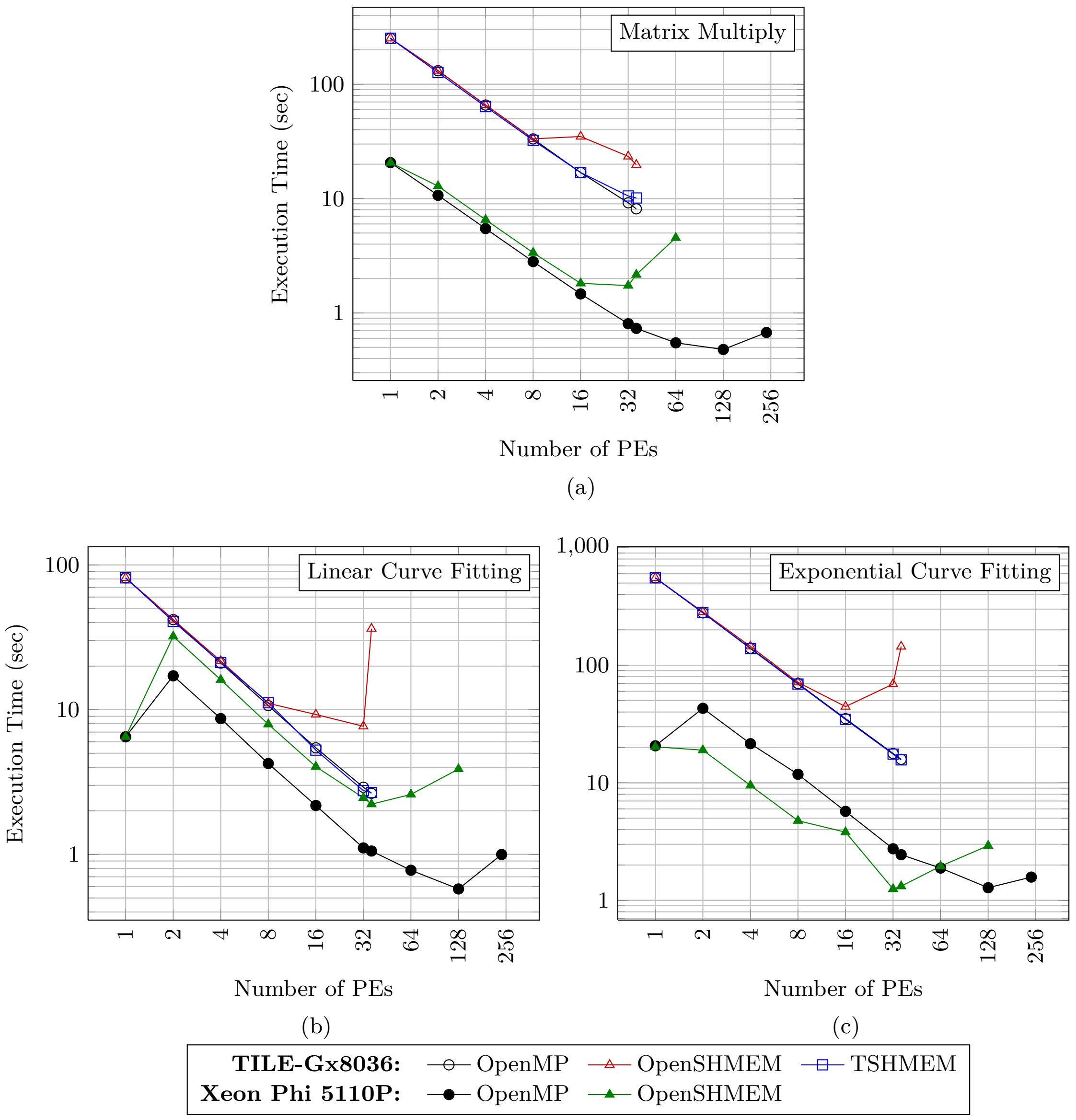

Fig. 1. Execution times for (a) matrix multiplication (2048×2048, double); (b) linear curve fitting (400M points, float); and (c) exponential curve fitting (400M points, float). For TILE-Gx, TSHMEM and OpenMP execution times commonly overlap.

Linear Curve Fitting. The second application is curve fitting via linear least-squares approximation. By using the least-mean-squares approximation, we can define a best-fit curve with minimal sum-of-squared deviation from an existing data set. If the data points are $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ where x and y are variables such that x is independent and y is dependent, then $f(x)$ can be defined as a curve with deviation d from each point in the data set. Least-squares approximation defines the best curve fit with the following deviation property:

$$\prod = d_1^2 + d_2^2 + \cdots + d_n^2 = \sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} [y_i - f(x_i)]^2 = \text{minimum}$$

For a straight line given by the equation $f(a, b) = a + bx$, the slope and intercept can be calculated as follows: $b = \sum(\mathrm{d}x\,\mathrm{d}y)/\sum \mathrm{d}x^2$; $a = \bar{y} - b\bar{x}$.

The implementation of linear curve fitting can be highly parallelized since the points distributed among the PEs can be used to compute the partial sum and the intermediate values of the deviations. These partial results can then be combined with summation reduction operations. This application provides a contrast between these two programming environments as the application is lightweight and primarily consists of memory-bound computation.

Figure 1b presents execution times for linear curve fitting for each programming environment and device. Similar to matrix multiplication on the TILE-Gx, the execution times of TSHMEM, OpenSHMEM, and OpenMP track each other until OpenSHMEM begins to stop scaling significantly after 8 PEs. Our TSHMEM library continues to outperform OpenSHMEM on TILE-Gx for all PE counts. The Xeon Phi performance shows similar scaling for OpenSHMEM and OpenMP, with a runtime disadvantage to OpenSHMEM. Interestingly, the serial baseline performs exceedingly well on the Xeon Phi without any parallelization due to aggressive cache prefetching, locality of reference across the unified L3 of the device, and strong memory performance for streaming data from arrays. The performance of this application begins increasing at 4 PEs for the OpenMP version and 8 PEs for the SHMEM version. OpenSHMEM on the Xeon Phi stops scaling after 36 PEs.

Exponential Curve Fitting. Our final application for SHMEM and OpenMP performance analysis on the TILE-Gx and Xeon Phi is curve fitting via exponential approximation. An exponential equation of the form $y = ae^{bx}$ can be represented in linear format via logarithm: $ln(y) = ln(a) + bx$. This form allows us to leverage the previous linear curve-fitting application and supplement it with logarithm functions for exponential approximation. Once this transformation is achieved, linear curve fitting is executed and the final result is transformed back by taking the inverse logarithm. Exponential curve fit is computationally intensive as compared to linear curve fit as it involves logarithmic transformation of points.

The execution times are presented in Figure 1c. For the TILE-Gx, the performance for OpenMP, OpenSHMEM, and TSHMEM follow the same pattern as the previous two applications. TSHMEM and OpenMP continue to track each other's execution times on the TILE-Gx, while OpenSHMEM does not scale as well with more than 16 PEs. The Xeon Phi, however, shows a surprisingly different trend. Unlike the two previous applications, OpenSHMEM outperforms the OpenMP version until it eventually stops scaling soon after 36 PEs. The performance difference was determined to be a log-deviation calculation whereby its reduction operation in OpenMP was seven times slower at 2 PEs than the same reduction with OpenSHMEM. Similar to linear curve fit, the serial baseline for exponential curve fit performs well when compared to OpenMP. OpenMP has parity performance with the serial baseline around 4 PEs. In comparison, OpenSHMEM is at parity performance with the serial baseline at 2 PEs. Both OpenMP and OpenSHMEM continue to increase speedup as PEs increase until 32 for OpenSHMEM and 128 for OpenMP.

3.3 SHMEM-only Applications

In conducting our analyses with SHMEM and OpenMP, writing applications to be algorithmically similar creates a disadvantage on optimization techniques available from each parallel-programming environment. While we pursued that approach in order to facilitate fair many-core platform comparisons for the computation and communication patterns of those algorithms, this subsection offers an alternative look. In this subsection, we present several parallel applications that were independently developed with the SHMEM programming library: matrix multiply and heat image from the OpenSHMEM test suite [11], a huge radix sort implementation, and process-based parallelization of the FFTW library with SHMEM. These applications are meant to showcase high performance with SHMEM as well as its specific tradeoffs.

OSH Matrix Multiply. The OpenSHMEM (OSH) website provides a test suite with benchmarks and applications [11]. One of the applications provided from the 1.0d test suite is a matrix multiplication kernel. The performance of this kernel is shown in Figure 2a for OpenSHMEM and TSHMEM for our many-core platforms. The serial baseline used is identical to the one from Figure 1a.

The execution times from Figures 1a and 2a show that OSH matrix multiplication is around 1.5 to 2 times slower than our own matrix multiplication presented in Section 3.2 when executed on the TILE-Gx. The Xeon Phi also performed worse for this application, reaching parity performance with the serial baseline at 16 PEs and stopped scaling soon after 36 PEs. The fundamental reason is due to the algorithm used in this implementation. As mentioned before, our matrix multiplication has to obtain a private copy of the second matrix, forcing the memory requirements to scale with the number of PEs and the size of the matrix. For large matrix sizes, these private copies become a problem as more and more PEs increasingly coexist on a single device. The communication time of obtaining this private copy also increases with more PEs in the system, explaining the slight increase in execution time at 32 and 36 PEs for TSHMEM compared to OpenMP in Figure 1a. The OSH matrix multiplication, however,

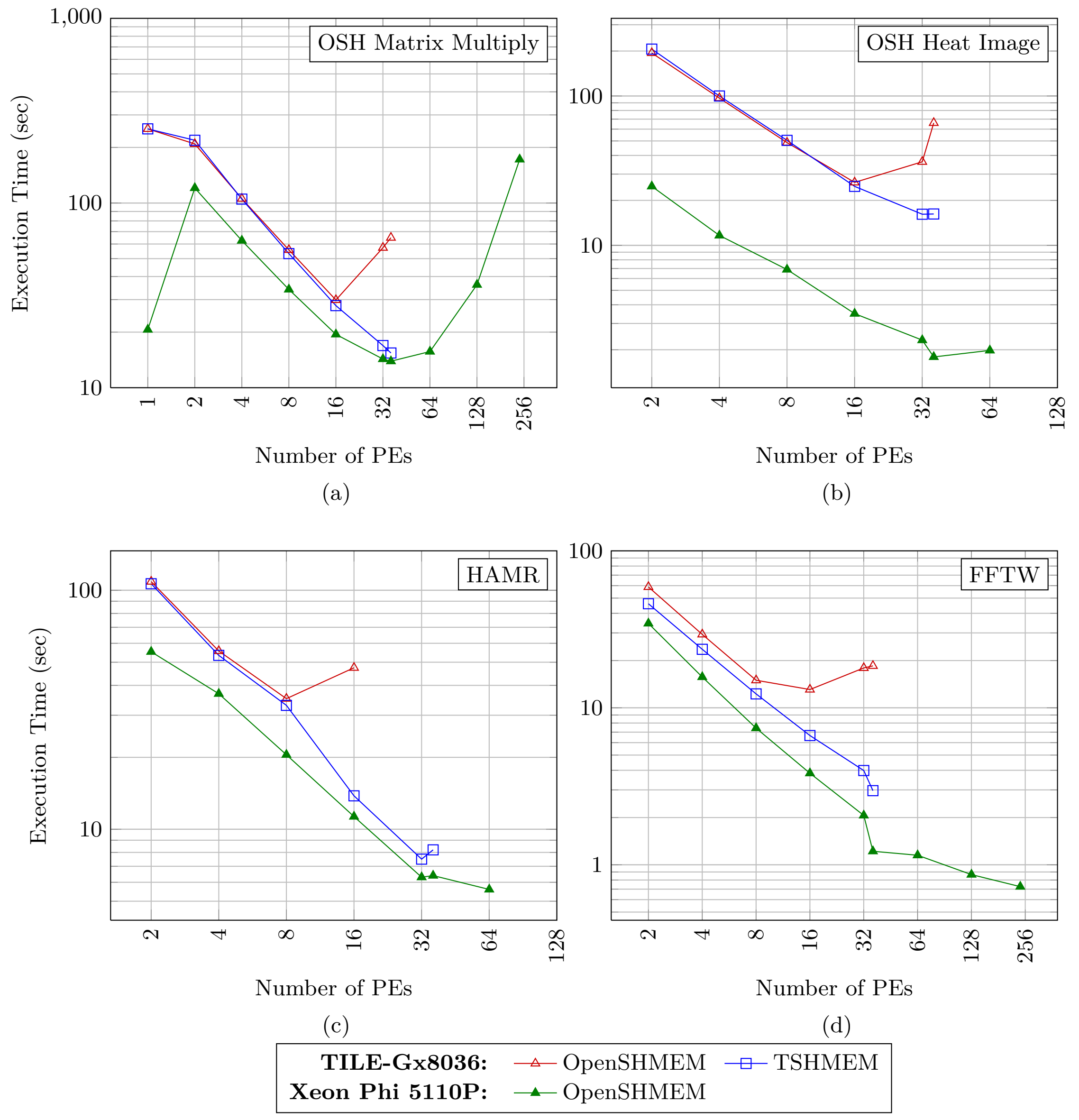

Fig. 2. Execution times for (a) OSH matrix multiply (2048×2048, double); (b) OSH heat image (1024×1024); (c) HAMR (3.8 GB); and (d) FFTW with SHMEM (8192 FFT operations on 8192-length float arrays).

uses a distributed data structure that divides up all matrices among the PEs. This data distribution results in more communication time to obtain non-local elements of the second matrix to perform matrix multiplication on its managed elements, but the advantage is substantially lower memory use for increasing PE sizes. While this approach is slower for these devices, it is more amenable for very large matrices, higher PE counts, or distributed systems with long latencies.

While the communication pattern plays a role in this application, TSHMEM on the TILE-Gx achieves the same runtime as OpenSHMEM on the Xeon Phi at or around 36 PEs. This result is worthy of mention due to the difference in power consumption of the two devices.

OSH Heat Image. The second application is also from the OpenSHMEM test suite: heat-conduction modeling. This application takes width and height parameters as inputs and solves a heat-conduction task. Each PE generates its own initial data, performs computation, reduction, and finally generates an output image.

Execution times are shown in Figure 2b. Trends in execution time show that OpenSHMEM stops scaling around 16 PEs on the TILE-Gx whereas TSHMEM continues on to effectively leverage the entire device. Due to the relatively sparse communication pattern in this application, the Xeon Phi executes an order of magnitude faster due to its more powerful cores compared to the TILE-Gx.

Huge, Asynchronous 128-bit MSB Radix Sort (HAMR). HAMR is an in-place huge radix sort. Designed as an application benchmark to test communication and integer comparisons, it fills the requested memory with random 128-bit integers and subdivides the work to the PEs. There are two main phases to the benchmark: (1) asynchronous parallel key exchange, and (2) local MSB radix sort. For any significant size of memory, the intensive all-to-all communication of the first phase contributes significantly more toward total runtime than the second phase.

Execution times for HAMR are shown in Figure 2c. OpenSHMEM on the TILE-Gx stops scaling around 8 PEs whereas TSHMEM continues and excels at communication. The Xeon Phi's OpenSHMEM also begins performing relatively poorly after 8 PEs as scaling becomes noticeably decreased compared to TSHMEM on the TILE-Gx. In addition to the communication, the TILE-Gx performs exceptionally well relative to the Xeon Phi for this benchmark. Excelling at integer operations, the TILE-Gx displays a higher computational density per watt of power given that the TDP of the TILE-Gx is around eight times less than the Xeon Phi's TDP.

Large, Distributed 1D-FFT with SHMEM and FFTW. The final application involves the process-based parallelization of a popular FFT library, FFTW [4]. The application performs a distributed 1D-DFT computation using the FFTW library, with data setup and inter-process communication via SHMEM.

While the FFTW library is already multithreaded internally, this application uses SHMEM instead of MPI to handle inter-process communication via fast one-sided puts to quickly exchange data. Designed primarily for cluster-based systems, we experiment with this application on our many-core platforms.

The execution times in Figure 2d show OpenSHMEM and TSHMEM executions on the TILE-Gx and Xeon Phi. The scalability of OpenSHMEM significantly decreases after 8 PEs on the TILE-Gx, but continues for the Xeon Phi. While previous applications showed slightly better performance with TSHMEM compared to OpenSHMEM at 8 PEs or less, execution times with FFTW show significantly superior performance for TSHMEM on the TILE-Gx. TSHMEM executions were approximately 20% faster than OpenSHMEM TILE-Gx executions. The weaker performance for OpenSHMEM with FFTW may have also manifested on the Xeon Phi, possibly explaining the close performance of TSHMEM on TILE-Gx to that of OpenSHMEM on the Xeon Phi.

3.4 Observational Experiences with SHMEM and OpenMP

To conclude our application analyses with OpenMP and SHMEM, we recap recurring themes and provide observational experiences with both of these programming environments.

In our experience with developing OpenMP and SHMEM applications, several advantages and disadvantages arose for both. OpenMP's main advantage is incremental parallelization of existing sequential applications. Through profiling, sequential code can be iterated on and gradually parallelized until the desired performance criteria are met. However, due to the use of compiler directives and developer-hidden parallelization, difficult-to-debug issues such as unnecessary synchronization, code serialization, or race conditions may occur. Additionally, OpenMP is typically unavailable for development on distributed systems and requires the use of other parallel-programming environments such as MPI for inter-node communication. The advantages of OpenMP's shared-memory model, however, allow it to remain a strong and viable choice for SMP-centric parallelization.

As alluded to previously, the OpenMP matrix multiplication application from Section 3.2 efficiently shares the three matrices with its child threads. When designing the same algorithm in SHMEM, sharing large data structures stored locally necessitates converting them into distributed data structures. For many-core devices, these distributed structures are not necessarily an advantage as the structures would still reside in the globally shared memory of SHMEM's symmetric PGAS, but incur more work for the developer to realize the same performance outcome for a single device. However, as shown from the results, SHMEM excels at large symmetric workloads whereby each PE receives computationally equivalent data and tasks. Asymmetric communication is also a strong suit for both SHMEM and OpenMP, but SHMEM performs them via standard library function calls. OpenMP, however, requires compiler support to implement these communication calls. As a result, SHMEM is highly suited for both local and distributed parallel programming (as was the original SHMEM on the Cray T3D distributed supercomputer) and is capable of taking the role of both MPI and OpenMP in a large, distributed, modern SMP-based cluster.

In all applications tested on the TILE-Gx, TSHMEM was able to successfully outperform the OpenSHMEM reference implementation. OpenSHMEM has scaling issues beyond 8 or 16 PEs, less than one-fourth or one-half of the device, and is partially attributed to less optimization for the TILE-Gx. Despite the underlying GASNet libraries cross-compiling successfully, the TILE-Gx is only generically supported. In contrast, OpenSHMEM on the Xeon Phi shows increased promise for scalability due to additional x86-based optimizations in the GASNet libraries, but several applications still display a surprisingly lower level of performance, especially relative to TSHMEM on the TILE-Gx (e.g., OpenSHMEM's matrix multiply, Figure 2a, 32 PEs). TSHMEM and OpenMP consistently scale together for these applications, demonstrating TSHMEM's potential for a hardware-aware bare-metal approach to designing a SHMEM library for many-core devices.

4 Conclusions

We have presented and evaluated three applications written in both OpenMP and SHMEM on the TILE-Gx and Xeon Phi. SHMEM implementations used include the OpenSHMEM reference implementation and our prior work with TSHMEM for Tilera many-core processors. In addition, four independently developed SHMEM applications were evaluated on our many-core platforms in order to emphasize comparative performance of the devices and the SHMEM-library implementations.

Several major contributions are illustrated with this work. The performance of SHMEM and OpenMP applications for our many-core platforms show scalability concerns for GASNet and the OpenSHMEM reference implementation on both platforms. When applicable, TSHMEM and OpenMP performance are comparable and scale similarly on the TILE-Gx. Furthermore, TSHMEM outperforms OpenSHMEM in execution times and scalability for all SHMEM applications evaluated. This conclusion provides validation to a bare-metal library design for TSHMEM on many-core devices.

Future work for this research includes power normalization of results to quantify each platform's computational density per watt, and additional results for the TILE-Gx and Xeon Phi with NAS Parallel Benchmarks [1] to evaluate their architectural strengths at different categories of computation and communication common among HPC applications.

Acknowledgements. This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant Nos. EEC-0642422 and IIP-1161022.

References

1. Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., Weeratunga, S.: The NAS Parallel Benchmarks. Tech. Rep. RNR-94-007, NASA Advanced Supercomputing Division (1994)
2. Bonachea, D.: GASNet specification, v1.1. Tech. rep., University of California at Berkeley, Berkeley, CA, USA (2002)
3. Dagum, L., Menon, R.: OpenMP: an industry standard API for shared-memory programming. Computational Science Engineering, IEEE 5(1), 46–55 (Jan-Mar 1998)
4. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proceedings of the IEEE 93(2), 216–231 (2005)
5. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing 22(6), 789–828 (1996)
6. Intel Corporation: Intel Xeon Phi coprocessor 5110P (2013), `http://ark.intel.com/products/71992/`
7. Lam, B.C., George, A.D., Lam, H.: TSHMEM: shared-memory parallel computing on Tilera many-core processors. In: Proc. of 18th International Workshop on High-Level Parallel Programming Models and Supportive Environments. HIPS '13, IEEE (2013)
8. Mellanox Technologies: Mellanox ScalableSHMEM (2013), `http://www.mellanox.com/related-docs/prod_software/PB_ScalableSHMEM.pdf`
9. Silicon Graphics International Corp.: SHMEM API for parallel programming (2013), `http://www.shmem.org/`
10. Tilera Corporation: TILE-Gx8036 processor family (2013), `http://www.tilera.com/products/processors/TILE-Gx_Family`
11. University of Houston: OpenSHMEM source releases (2013), `http://openshmem.org/site/Downloads/Source`