An Introduction to TSHMEM for Shared-Memory Parallel Computing on Tilera Many-Core Processors

Bryant C. Lam Alan D. George Herman Lam
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611-6200
{blam, george, hlam}@chrec.org

1. INTRODUCTION

Parallel programming is experiencing explosive growth of demand due to processor architectures shifting toward many processing cores in an effort to maintain performance progression in the face of technological and physical limitations. With the emergence of many-core processors into high-performance computing (HPC), the development of parallel programming models, tools, and libraries is more essential than ever before.

We briefly present and evaluate the design of TSHMEM (TileSHMEM), a SHMEM library based on the OpenSHMEM version 1.0 specification [1]. TSHMEM delivers a high-performance many-core programming library that improves developer productivity 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 with microbenchmarking employed in order to compare the realizable performance and overhead between those libraries and TSHMEM functionality.

Our research focuses on the TILE-Gx36 with its predecessor, the TILE*Pro*64, as baseline. The TILE*Pro* is Tilera's previous generation of many-core processors with 32-bit processing cores interconnected via four dynamically dimension-order-routed networks and one developer-defined statically routed network. The TILE-Gx is Tilera's new generation of 64-bit many-core processors. Each 64-bit processing core is now attached to five dynamic networks. In addition, TILE-Gx provides 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. DEVICE PERFORMANCE STUDIES

Tilera provides the Tilera Multicore Components (TMC) library for general application development, suitable for any task model and componentized such that developers only leverage the routines needed. In addition, the gxio library provides programmability for features specific to TILE-Gx devices, such as mPIPE and MiCA. Benchmarking these libraries is necessary to determine the upper bound on performance realizable for any library design (e.g., TSHMEM) or application. Routines relevant to the functionality required in TSHMEM are microbenchmarked to compare performance and overhead. These include TMC common memory for shared-memory programming, TMC spin and sync barriers for processing synchronization, and UDN helper functions that communicate explicitly on the iMesh.

TSHMEM leverages common memory to provide the PGAS model and shared-memory semantics of SHMEM. The bandwidth of memory-copy operations to and from this shared memory is decisively important in TSHMEM due to significant use in one-sided data transfers. Effective bandwidth on TILE-Gx36 experiences three significant transitions in performance. The first two transitions are attributed to and occur at the L1d (32 kB) and L2 (256 kB) cache sizes. The L1d cache performance tops out around 3100 MB/s, and the L2 cache performance reaches a peak between 1900 MB/s and 2700 MB/s. Tilera's DDC (Dynamic Distributed Cache) is attributed to the third performance transition. DDC presents a large L3 unified cache that is the summation of each tile's L2 cache. Effective bandwidth decreases from 1000 MB/s as transfer sizes beyond 1 MB begin exceeding the L2 caches of nearby tiles from DDC, converging at 320 MB/s in memory-to-memory transfers. The TILE*Pro*64 follows the same trends experienced with TILE-Gx36. Performance is stable at or near 500 MB/s through the L1d and L2 cache sizes and decreases into memory-to-memory transfers (370 MB/s).

The TMC library provides two types of barriers for synchronization: spin and sync. As expected, spin barriers vastly outperform sync barriers due to their polling nature, with latencies of 1.5 µs and 47.2 µs at 36 tiles for the TILE-Gx36 and TILE*Pro*64, respectively, compared to 321 µs and 786 µs. Futhermore, the spin barrier for the TILE-Gx significantly outperforms the TILE*Pro*'s.

Tilera provides access to the UDN (User Dynamic Network), a low-latency direction-order-routed dynamic network on their iMesh. The TMC library provides UDN helper routines that facilitate these transfers via two-sided send-and-receive calls. Minimum payload (8 bytes for TILE-Gx, 4 bytes for TILE*Pro*) throughput for neighbor-to-neighbor, side-to-side, and corner-to-corner cases between two tiles on the UDN is 2900, 2500, and 2000 Mbps on TILE-Gx and 1700, 1300, and 980 Mbps on TILE*Pro*.

3. DESIGN OVERVIEW OF TSHMEM

The SHMEM communication library [2] 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

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

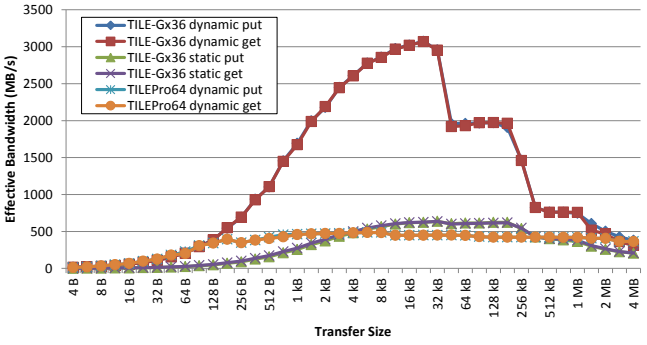

Figure 1: Effective bandwidth of TSHMEM put/get transfers for dynamic-dynamic on TILE-Gx36 and TILE*Pro*64 and static-static on TILE-Gx36.

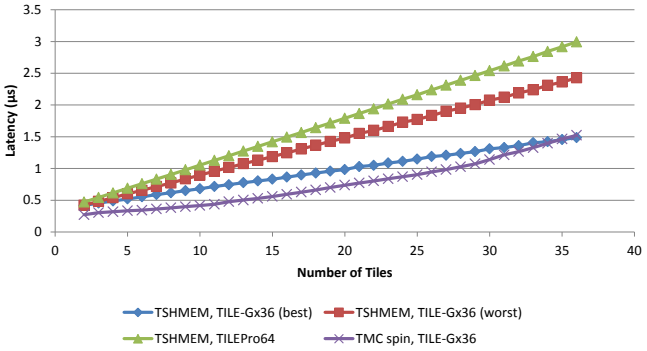

Figure 2: Latencies of TSHMEM barrier.

symmetric partition consists of symmetric objects (variables or arrays) of the same size, type, and relative address on all PEs. TSHMEM currently supports a subset of the OpenSHMEM V1.0 specification and implements the core functions commonly required by most SHMEM applications. These functions include symmetric memory management, one-sided put/get transfers, barriers, memory fencing, point-to-point sync, and collectives (e.g., broadcast, reduction). Atomic operations are not currently available in TSHMEM.

3.1 Point-to-Point Data Transfers

OpenSHMEM specifies several categories of point-to-point data transfers consisting of elemental, bulk, and strided put/get operations.

Dynamic symmetric variables are allocated at runtime on all PEs via SHMEM's dynamic memory allocation function `shmalloc()`. Due to the symmetry of each partition, a tile can determine the virtual address of any other tile's dynamic symmetric object by calculating the offset of its own object from its partition's start address and then adding the offset to the target tile's partition start address.

Static variables reside in the heap segment of the program executable and are allocated during link time. Unfortunately, the heap space resides in private memory of a process and is not directly accessible to other processes. TSHMEM facilitates data transfer from these static symmetric objects via UDN interrupts on Tilera's iMesh interconnect. If the local tile cannot service an operation, it can notify the remote tile over UDN for assistance. In the case when both target and source addresses point to static symmetric variables, neither local or remote tile will be able to service the operation. One of the tiles will create a temporary shared-memory buffer to assist in the transfer, incurring an additional memory copy operation as overhead. Static symmetric variable transfers in TSHMEM are not currently supported on the TILE*Pro* architecture due to lack of support for UDN interrupts.

Figure 1 shows the effective bandwidth for dynamic and static put/get transfers in TSHMEM. Note that put performance closely aligns with get performance for both the TILE-Gx36 and TILE*Pro*64. TSHMEM dynamic put/get operations demonstrate low overhead in comparison with the performance of the common memory microbenchmark. Major performance penalty is incurred only in the static-static case when temporary shared-space allocation is required to aid in the transfer.

3.2 Synchronization

OpenSHMEM specifies several categories of synchronization: barrier sync; communication sync with fence/quiet; and point-to-point sync (waiting until a variable's value changes).

Microbenchmark results for Tilera's TMC spin and sync barriers illustrate that using sync barriers is not feasible due to very high latency and the spin barrier on TILE*Pro* is significantly slower than the one on TILE-Gx. Consequently, TSHMEM implements OpenSHMEM barriers using the UDN to synchronize between tiles. The start tile in the active set generates an active-set identification for the barrier in order to prevent overlapping barrier calls from returning out-of-order or stalling. The active-set identification is encoded with a *wait* signal and is sent to the next tile and resent linearly until the last tile sends it back to the start, acknowledging that all participating tiles have reached the same execution point in the program. The process is then repeated with a *release* signal, allowing the blocking processes to linearly forward the signal before resuming program execution.

The performance of TSHMEM barriers is shown in Figure 2. As noted, the TILE*Pro*64's TSHMEM barrier with a latency of 3 µs at 36 tiles vastly outperforms its corresponding TMC spin barrier (47.2 µs). Due to a higher clock frequency, the TILE-Gx36's TSHMEM barrier exhibits lower latencies of 1.5 to 2.4 µs at 36 tiles.

4. CONCLUSIONS AND FUTURE WORK

We have briefly presented and evaluated our software architecture and design for TSHMEM, a high-performance OpenSHMEM library built atop Tilera-provided libraries for their Tilera TILE-Gx and TILE*Pro* many-core architectures. Performance is demonstrated with microbenchmarks of Tilera-library and TSHMEM functions, offering direct validation of realizable performance and any inherited overhead. Future work of TSHMEM will focus on fully achieving OpenSHMEM compliance via addition of atomic operations. We also plan to leverage novel architectural features of the TILE-Gx such as the mPIPE packet engine as we explore designs for expanded functionality in TSHMEM across multiple many-core devices.

5. REFERENCES

[1] OpenSHMEM. OpenSHMEM API, v1.0 final, 2012. `http://www.openshmem.org/`.

[2] Silicon Graphics International Corp. SHMEM API for parallel programming, 2012. `http://www.shmem.org/`.