Experiences with UPC on TILE-64 Processor

Olivier Serres, Ahmad Anbar, Saumil Merchant and Tarek El-Ghazawi
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Dept. of Electrical and Computer Engineering
The George Washington University, 801 22nd St NW, Washington DC 20052, USA
{serres, anbar}@gwmail.gwu.edu, {smerchan, tarek}@gwu.edu

Abstract—Partitioned global address space (PGAS) programming model presents programmers with a globally shared address space with locality awareness and one-sided communication constructs. The shared address space and the one-sided communication constructs enhance ease-of-use of PGAS based languages and the locality awareness enables programmers and the runtime systems to achieve higher performance. Thus PGAS programming model may help address the escalating software complexity issues resulting from the proliferation of many-core processor architectures in aerospace and computing systems in general. This paper presents our experiences with Unified parallel C (UPC), a PGAS language, on the Tile64™ processor, a 64-core processor from Tilera Corporation. We ported Berkeley UPC compiler and runtime system on the Tilera architecture and evaluated two separate runtime implementation conduits of the underlying GASNet communication library, a pThreads based conduit and an MPI based conduit. Each conduit uses different on-chip, inter-core communication networks providing different latencies and bandwidths for inter-process communications. The paper presents the implementation details and empirical analyses of both approaches by comparing and evaluating results from NAS Parallel Benchmark suite. The analyses reveal various optimization opportunities based on specific many-core architectural features which are also discussed in the paper[12].

TABLE OF CONTENTS

1. INTRODUCTION

The performance trend of microprocessors exploiting faster clock speeds and instruction-level parallelism has subsided due to unsustainable thermal and power overheads. The continued quest for higher performance has initiated a new focus, emphasizing thread- and task-level parallelism, leading to the emergence of homogeneous and heterogeneous multi- and many-core processors. But this trend lends the responsibility of achieving higher performance on the shoulders of the mainstream programmers. A programmer must now understand and exploit parallelism, in the past a specialized activity performed by a select few from the HPC community. To exploit the full potential of these new multi-core processors new design methodologies and languages are needed that can abstract low-level details, but at the same time enable the mainstream programmer to extract thread and task-level parallelism from the application. HPC community has over the last few decades extensively researched parallel design languages and methods, results and lessons from which form a strong foundation from which to develop new methodologies amenable to the mainstream programming community. A relatively newer design paradigm that has influenced many new HPC languages is the Partitioned global address space (PGAS) model. PGAS offers a global, logically shared memory space for all the threads, with locality awareness and one-sided communication constructs. It offers advantages on two fronts: (i) performance, and (ii) ease of use, significantly enhancing user productivity. Data-locality awareness resulting from partitioned address space enables the programmer to exploit locality information for higher performance. Global, shared memory logical view and one-sided communication constructs lend higher programmability enabling ease-of-use. Further, PGAS languages are nearly ubiquitous enhancing code portability. Several existing and upcoming parallel programming languages such as the Unified Parallel C (UPC), Co-Array Fortran (CAF), Titanium, Chapel, and X10 support the PGAS programming paradigm.

This paper presents our experiences on porting and evaluating UPC compiler and runtime system on the Tile 64 processor, a 64-core processor from Tilera [1]. The paper presents the analyses of our implementation approaches based on a set of standard application benchmarks. The paper is organized as follows. Section 2 presents a brief overview of the UPC language. Section 3 discusses the architecture of the Tile64 processor. Section 4 presents related work. Section 5 describes the UPC compilation flow on the Tile64 processor. Section 6 discusses the benchmark results and their evaluation. Concluding remarks are presented in section 7.

[1] 978-1-4244-7351-9/11/$26.00 ©2011 IEEE
[2] IEEEAC paper#1609, Version 4, Updated 2011:01:08

2. Overview of Unified Parallel C

UPC is an explicit parallel extension of the C language supporting the PGAS programming paradigm [4]. It offers common C language syntax familiar to the domain scientists that form the bulk of the HPC user community, thus easing the learning curve and providing a high productivity development environment [5]. The UPC execution model supports Single Program, Multiple Data (SPMD) styled parallel programming where a specified number of threads execute independently in parallel on multiple processors. It provides various synchronization mechanisms for the threads such as barriers, locks, and memory consistency control statements. UPC memory model supports and extends the PGAS memory model concepts. As in PGAS, the UPC memory model provides a global, shared memory space partitioned to provide affinity for portions of the shared address space resident locally on the host processor system. In addition, it also provides a private memory for each thread not accessible to other threads for local computations. Variable declarations can explicitly state shared versus private storage space. A pointer-to-shared variable declaration can reference all locations in the shared space. A private pointer can only reference the addresses in the thread private address space or its local portion of the shared space. Thus, syntactically both remote and private memory accesses are simple variable assignment statements, providing a standard logical abstraction for the one-sided remote communications. Both the static and dynamic memory allocations are supported for shared and private memory addresses. Figure 1 shows the UPC memory model.

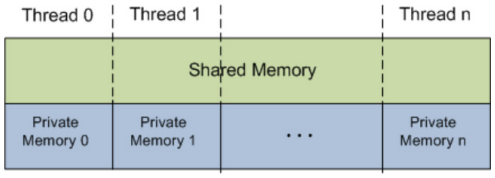


Figure 1. UPC memory model

3. Tile64 Processor Architecture

The Tile64 processor features 64 identical processor cores (tiles) interconnected with Tilera's iMesh on-chip network [2,3]. Each tile consists of a complete, full-featured 3-way VLIW processor as well as L1/L2 caches and a non-blocking switch that connect the tiles into the mesh. Four on chip memory controllers connect the tiles to on-board DDR memories. The iMesh interconnect offers multiple networks with different latencies and bandwidths for inter-tile, memory, and I/O communications. The iMesh interconnect offers five on-chip internal networks, each full duplex and 32-bits wide.

- *Static Network (STN)*: Static, scalar network with low latency allowing static configuration of the routing decisions. It is mainly used for streaming data from one tile to another via pre-configured routes.
- *User Dynamic Network (UDN)*: Dynamic, low latency, user programmable, packet switched network used for communications between threads running in parallel on multiple tiles.
- *Tile Dynamic Network (TDN):* Dynamic network, supports data transfer between tile caches. TDN works in concert with MDN.
- *Memory Dynamic Network (MDN):* Dynamic network used for memory transfers such as loads, stores, and cache misses.
- *Input/Output Dynamic Network (IDN):* Dynamic network accessible to operating system (OS)-level code not user applications. Used primarily for transfers between tiles and I/O devices, and I/O devices and memory.

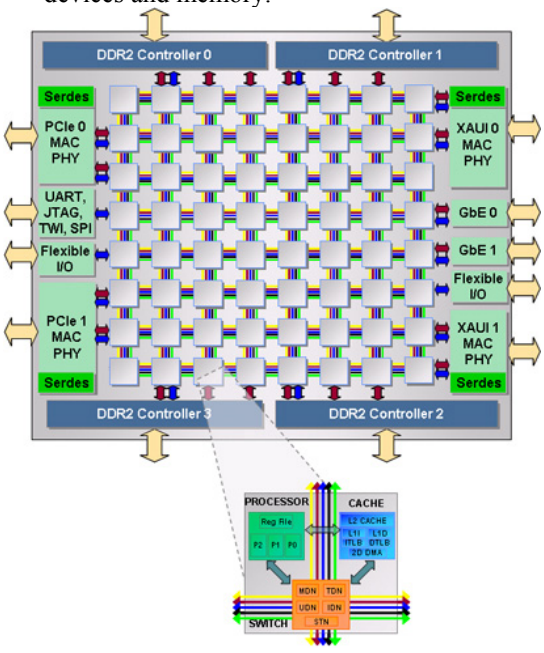


Figure 2. Tile64 architecture

Figure 2 shows the architecture diagram of the Tile64 processor. Each tile can independently run a full operating system, or, multiple tiles taken together can run a multi-processor operating system like an SMP version of GNU/Linux. The Tile64 architecture exhibits many interesting features which can be taken advantage of via the PGAS programming model. The chip provides physically shared DDR2 memory across all the tiles. This can aid in significantly reducing the shared memory abstraction overhead of the PGAS languages which otherwise is a dominant factor on the distributed memory systems. The variety of on-chip mesh networks can facilitate optimizing the synchronization primitives and the collectives. They can also optimize implementations of remote memory accesses and active messages. Just as the architectural features of the Tile64 can facilitate the PGAS model, so can PGAS

features such as locality awareness exhibited by the UPC language aid in exploiting performance on the manycore architectures. The locality information can especially come in handy for the exploiting higher performance on Tile64 architecture due to its small cache sizes.

4. Review Of Related Work

There have been few research efforts evaluating PGAS programming model for many-core architectures. To the best of our knowledge none have looked at targeting UPC on many-core processors, especially the Tile64 processor.

Underwood et al. evaluated hardware support in existing NIC technologies for the PGAS programming model [6]. In their work, they focused on the communication between nodes rather than on-chip inter-core communications. Santhanaraman et al. used special communication primitives offered by InfiniBand networks to implement and evaluate one-sided communication constructs in parallel programming models [7]. They used MPI for their evaluation and took advantage of one-sided atomic operations on cache-coherent multi-core/multi-processor architectures while still exploiting the benefits of networks such as InfiniBand.

For many years, message-passing interface MPI [8] has been considered the de-facto standard for high performance computing programs. MPI follows the message-passing programming model. But the added programming complexity with MPI makes it less suitable for mainstream computing on many-core architectures. In [9], Krawezik showed that OpenMP [10], which follows the shared memory programming model, is giving almost the same performance as MPI but without the need to pay the price of strong programming effort.

In [11], a comparison is made between the performances of UPC, a PGAS language, to those of MPI. It is shown that UPC and MPI equally performed when the program communication patterns rely on large messages. At the same time it also showed that UPC out-performed MPI when relying on mid and small size messages. Performance and productivity analyses of PGAS languages have been presented in many paper [5, 12, 13, 14]. [5] shows that UPC has consistent improvement over MPI in terms of number of lines, number of characters and conceptual effort. The Tile 64 processor interconnection architecture was presented in [2]. This paper also included some microbenchmarks results for bandwidth and latency of the Tile64 processor on-chip networks.

5. UPC Compilation Flow

We ported the Berkeley UPC compiler and runtime system to the Tile64 processor. Figure 3 shows the UPC compilation flow. The compilation uses the Berkley UPC-to-C translator [15] and the Tilera C compiler [1] to compile UPC source codes. The Berkley UPC-to-C translator performs a source-to-source translation from a UPC code to an ANSI C compliant code with shared memory operations transformed into calls to the UPC runtime system built over the Global Address Space Networking (GASNet) communication infrastructure. GASNet is a language independent, low-level networking layer that provides high-performance communication primitives tailored for implementing PGAS languages such as UPC [16]. GASNet is partitioned into two layers to maximize portability without sacrificing performance. The GASNet core API is the low-level, network architecture dependent layer based heavily on active messages. The GASNet extended API is built on top of the core API and provides higher-level operations such as remote memory access and collective operations.

Our design flow uses two separate GASNet implementations on the Tile64 processor. The first uses a pThreads based GASNet conduit built using the Linux's pThread library and the second uses an MPI based GASNet conduit. The pThreads based GASNet conduit uses the memory dynamic network (MDN) on the Tile64 processor for implementing remote accesses whereas the MPI based conduit uses the user dynamic network (UDN). This is illustrated in Figure 4. The ported compiler and runtime implementations were thoroughly verified using the GWU Unified Testing Suite (GUTS) for UPC compilers [17].

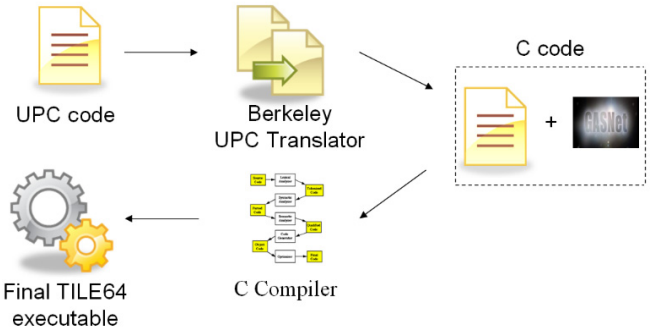


Figure 3. UPC compilation flow

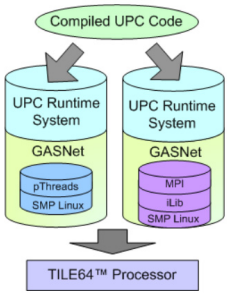


Figure 4. Berkeley UPC system implementations

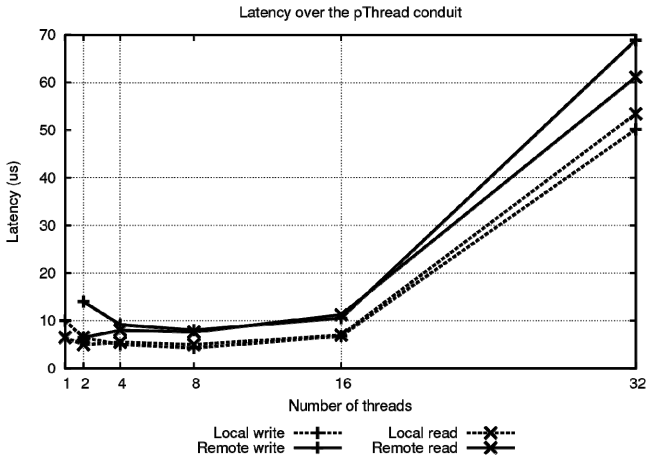


Figure 5. Latency over the pThread conduit

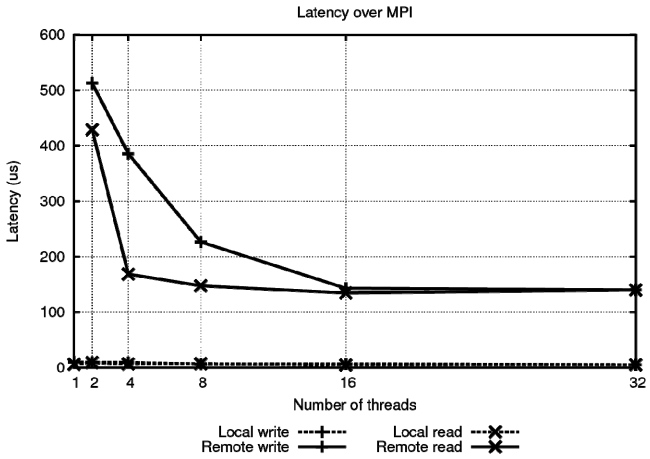


Figure 6. Latency over the MPI conduit

6. PERFORMANCE EVALUATION

The performance evaluation of the two conduits was conducted using both micro-benchmarks and more classical full-featured benchmarks commonly used to benchmark supercomputers.

6.1 Microbenchmarks

A set of simple micro-benchmarks have been implemented to evaluate the implementations. The micro-benchmarks reveal the memory communication performance and bottlenecks of the two approaches. Memory access latencies are shown in figures 5 and 6. The bandwidth for local reads/writes (targeting the private memory of the thread) and remote reads/writes (targeting other threads shared memory) are shown for transfers of 4kB in figures 7 and 8, for 256 kB in figures 9 and 10, and for 512 kB in figures 11 and 12.

6.2 Application Benchmarks

Kernels of the NAS parallel benchmarks (NPB) [18, 19], version 2.4 have been used to compare the performances of the two approaches. The NPB are commonly used to benchmark high-performance computer systems. The UPC versions of those benchmarks were presented in [20]. They are chosen due to their wide variety of memory access patterns. The results shown here are for dataset size NPB class A. NPB MultiGrid (MG) solves a 3D Poisson equation using a V-cycle multigrid method; it exhibits structured, long range communications. NPB Conjugate Gradient (CG) computes the smallest eigenvalue of a matrix; it uses irregular, long range communications. NPB Embarrassingly Parallel (EP) generates pairs of Gaussian random deviates, nearly no communication is used. NPB Integer Sort (IS) performs a bucket sort on small integers. Finally, NPB Fourier Transform (FT) is a 3D fast Fourier transform benchmark stressing global communications.

Analyses of the results reveal many optimization opportunities. Microbenchmark results show lower remote memory access latencies for the pThreads approach (refer figures 5 and 6), but the bandwidth cannot scale well beyond four threads impacting its performance on most benchmark applications (see figures 7 through 12). The MPI based approach exhibits good scalability on most of the application benchmarks, even though the latencies with the MPI approach are much higher as compared to the pThreads approach.

The local bandwidth performances are also slightly better and less dependent of the number of threads with the MPI conduit. This is mostly due to the fact that MPI processes are not moved by the OS between tiles.

The NAS Parallel Benchmark kernels provide good performance and scalability for both conduits (Figures 13 through 22); the speedup figures represent the speedup over the best performing conduit with one thread. NPB IS is particularly suitable for the Tile64 for its exclusive usage of integer operations. The pthread conduit provides better results due to its lower latency, except for NAS EP which is not much sensible to the latency. It is also important to note that there are still many optimization opportunities on the Tile platform as the MPI conduit is built over many layers which all add overheads: GASNet, MPI, iLib, and Linux.

Also under investigation are ways to fully utilize the on-chip iMesh networks to optimize the GASNet implementation on Tile64 processor. A hybrid runtime using both the UDN and the memory management networks of the Tile64 would be able to provide both low-latency for small reads and writes but also high-bandwidth for bigger packets. The collective and the synchronization operations could also take advantages of the STatic Network (STN).

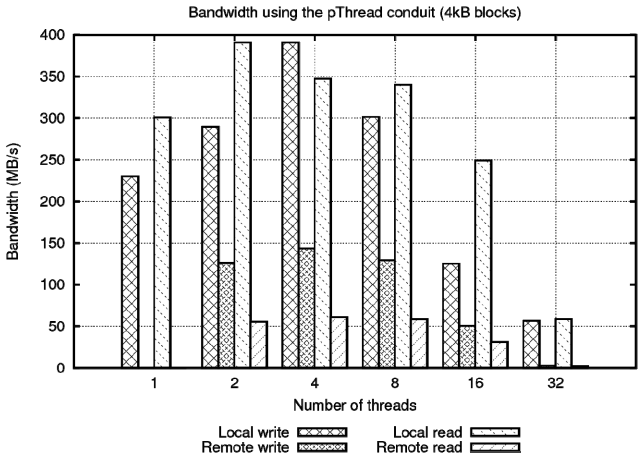


Figure 7. BW using the pThread conduit (4kB blocks)

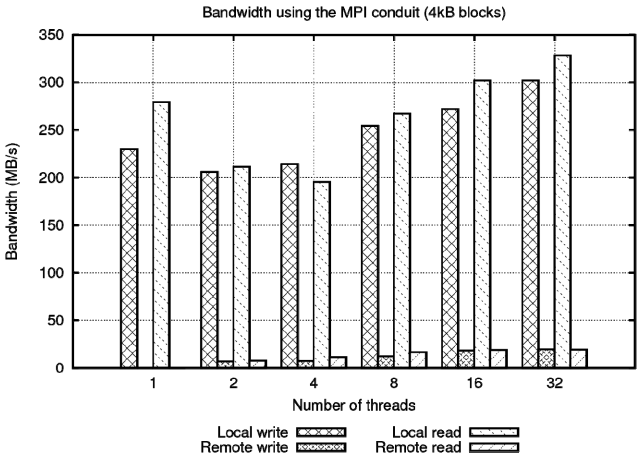


Figure 10. BW using MPI conduit (4kB blocks)

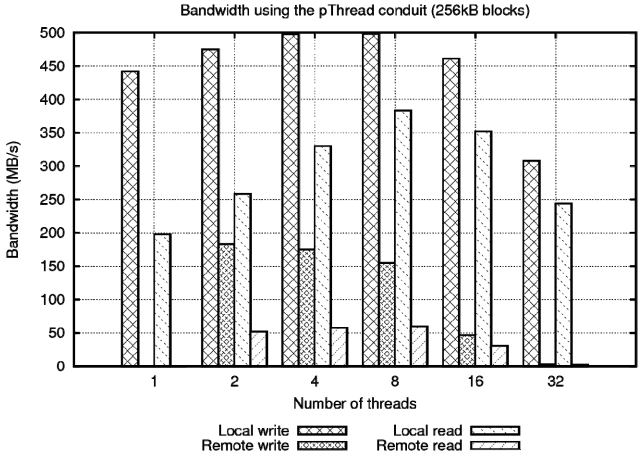


Figure 8. BW using the pThread conduit (256kB blocks)

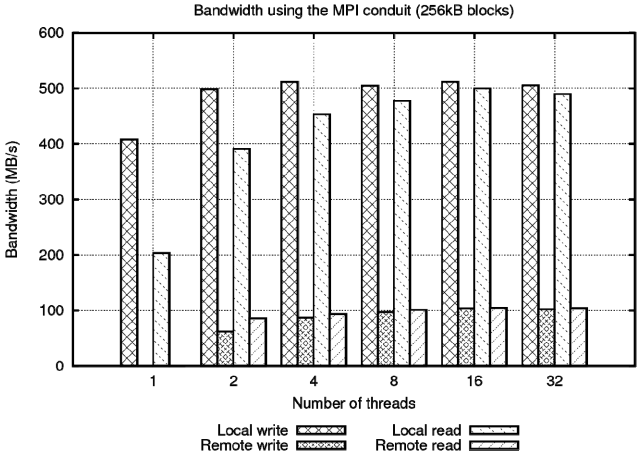


Figure 11. BW using MPI conduit (256kB blocks)

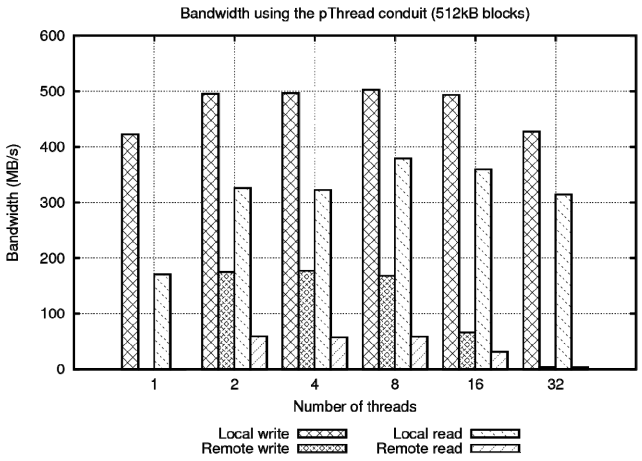


Figure 9. BW using pThread conduit (512kb blocks)

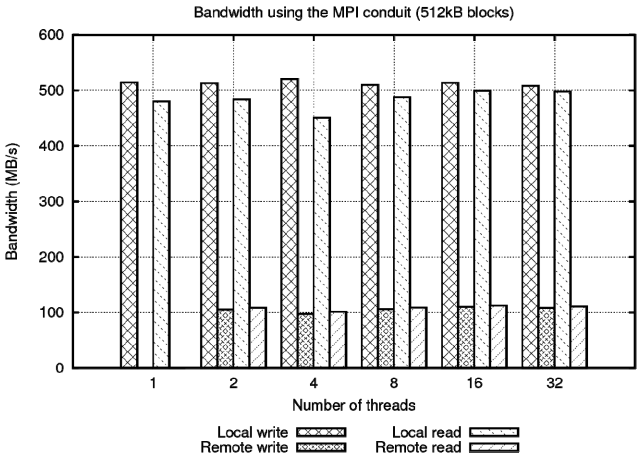


Figure 12. BW using MPI conduit (512kB blocks)

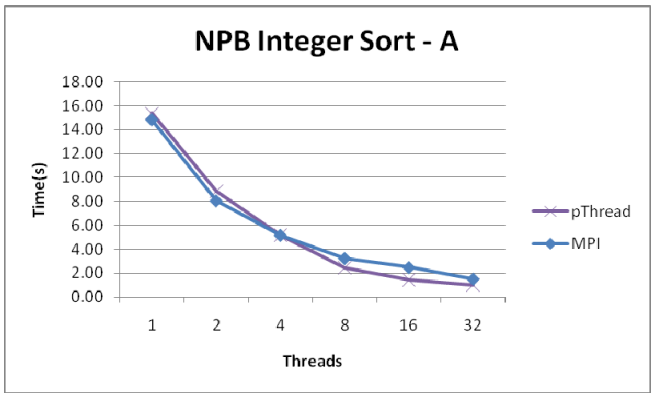


Figure 13. NPB IS (Execution time)

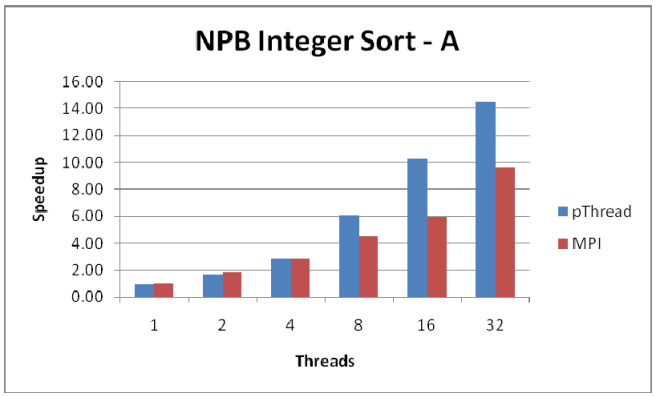


Figure 14. NPB IS (Speedup)

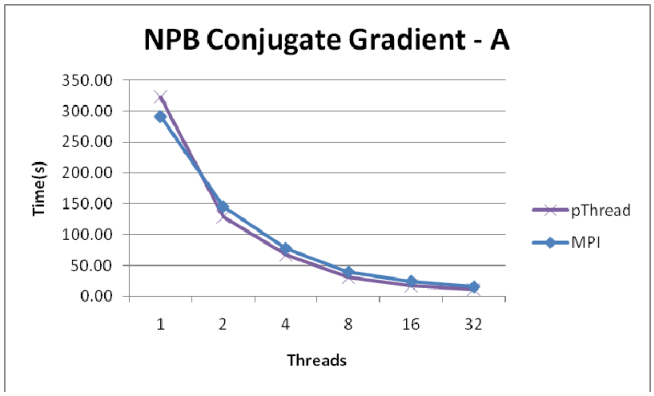


Figure 15. NPB CG (Execution time)

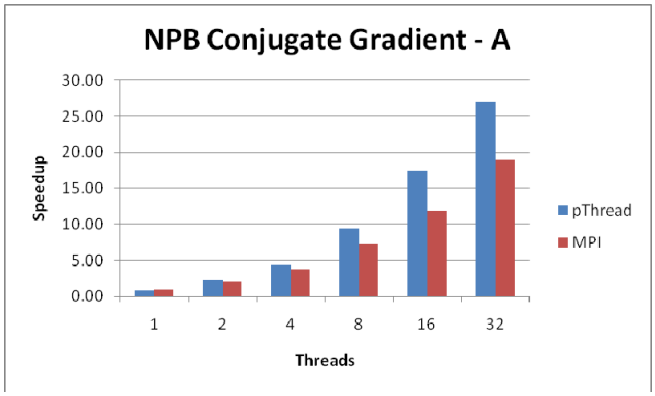


Figure 16. NPB CG (Speedup)

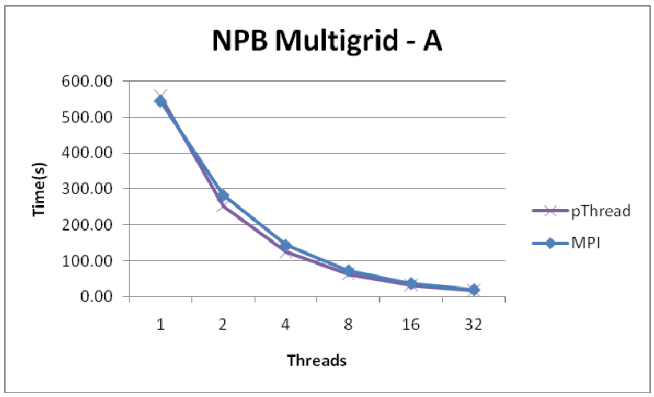


Figure 17. NPB Multigrid (Execution Time)

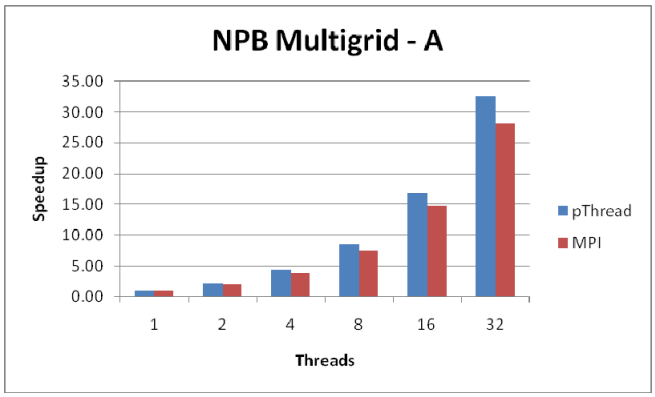


Figure 18. NPB MG (Speedup)

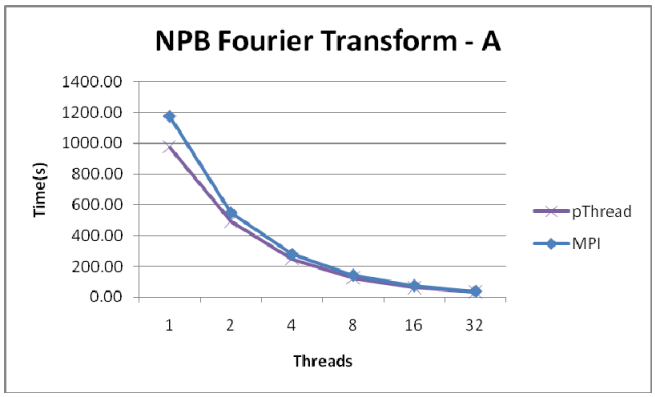


Figure 19. NPB FT (Execution time)

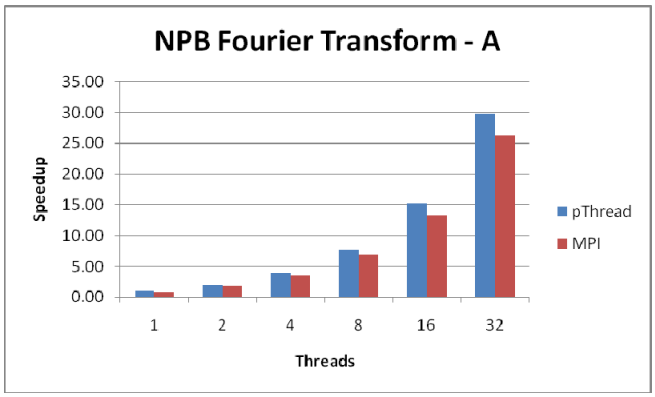


Figure 20. NPB FT (Speedup)

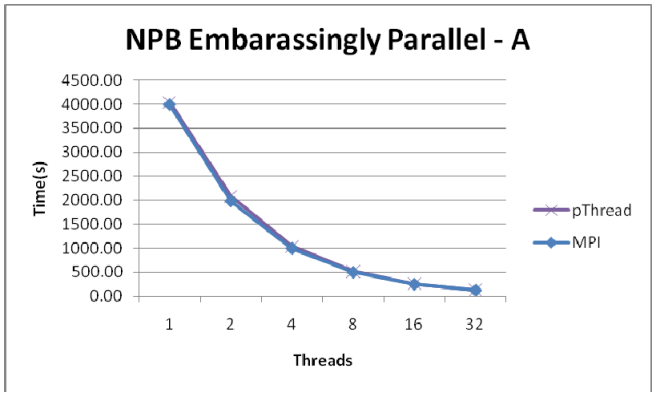


Figure 21. NPB EP (Execution time)

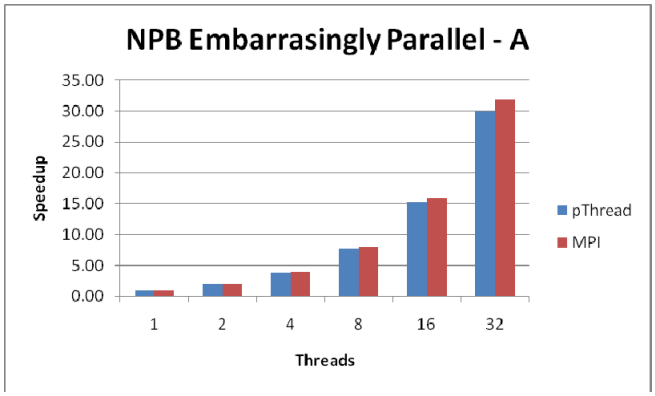


Figure 22. NPB EP (Speedup)

7. CONCLUSIONS

PGAS languages are gaining prominence amongst the HPC user community. One-sided communication mechanisms, global shared memory abstraction with data locality awareness, and simple programming constructs in these languages provide increased programmability and performance. This paper presents UPC runtime system implementations on the Tile64 processor. The design flow uses the Berkley UPC-to-C translator and the Tilera C compiler to compile UPC code. The UPC runtime system is built on top of the GASNet networking infrastructure implemented using two approaches: (i) the pThreads based GASNet conduit, and (ii) the MPI based GASNet conduit. Analyses of the benchmark results reveal several optimization opportunities, especially targeted for manycore architectures. These include use of physically shared memory to reduce address translation overheads associated with PGAS languages, optimal use of iMesh interconnects for faster synchronization and collectives, and optimized thread placement strategies for efficient use of available resources. These are currently under study. The objective of this work is to design a highly optimized UPC runtime system implementation built over GASNet for the Tile64. The end goal of this project is to investigate PGAS programming model benefits on many-core architectures. This will enable us to identify future architectural improvements in many-core technology to better support PGAS languages and study PGAS model extensions to better exploit many-core architectural features.

ACKNOWLEDGEMENT

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. IIP-0706352.

REFERENCES

[1] Tilera Corporation, “www.tilera.com.”

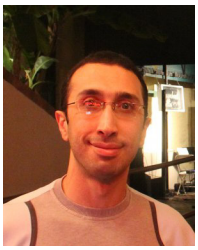
[2] D.Wentzlaff, P. Griffin, H. Hoffman, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal, “On-chip interconnection architecture of the tile processor,” Micro, IEEE, vol. 27, no. 5, pp. 15–31, Sept.-Oct. 2007.

[3] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C. C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, “Tile64 - processor: A 64-core soc with mesh interconnect,” Feb. 2008, pp. 88–598.

[4] The UPC Consortium, “UPC language specifications v1.2 (www.gwu.edu/˜upc/docs/upc_specs_1.2.pdf),” 2005.
[5] F. Cantonnet, Y. Yao, M. Zahran, and T. El-Ghazawi, “Productivity analysis of the UPC language,” April 2004, pp. 254.
[6] K. D. Underwood, M. J. Levenhagen, and R. Brightwell, “Evaluating NIC hardware requirements to achieve high message rate PGAS support on multi-core processors,” in SC ’07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing. New York, NY, USA: ACM, 2007, pp. 1–10.
[7] G. Santhanaraman, P. Balaji, K. Gopalakrishnan, R. Thakur, W. Gropp, and D. Panda, “Natively supporting true one-sided communication in MPI on multi-core systems with infiniband,” May 2009, pp. 380–387.
[8] MPI Forum, “MPI: A message-passing interface standard,” Knoxville, TN, USA, Tech. Rep., 1994.
[9] G. Krawezik, “Performance comparison of MPI and three OpenMP programming styles on shared memory multiprocessors,” in SPAA ’03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures. New York, NY, USA: ACM, 2003, pp. 118–127.
[10] OpenMP Forum, “OpenMP C and C++ application program interface, version 1.0. http://www.openmp.org,” October 1998.
[12] T. El-Ghazawi, F. Cantonnet, Y. Yao, S. Annareddy, and A. S Mohamed, “Benchmarking parallel compilers: a UPC case study,” Future Gener. Comput. Syst., vol. 22, no. 7, pp. 764–775, 2006.
[13] C. Coarfa, Y. Dotsenko, J. Mellor-Crummey, F. Cantonnet, T. El-Ghazawi, A. Mohanti, Y. Yao, and D. Chavarría-Miranda, “An evaluation of global address space languages co-array Fortran and Unified Parallel C,” in PPoPP ’05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming
[14] K. Yelick, D. Bonachea, W.-Y. Chen, P. Colella, K. Datta, J. Duell, S. L. Graham, P. Hargrove, P. Hilfinger, P. Husbands, C. Iancu, A. Kamil, R. Nishtala, J. Su, M. Welcome and T. Wen, “Productivity and performance using partitioned global address space languages,” in PASCO ’07: Proceedings of the 2007 international workshop on Parallel symbolic computation. New York, NY, USA: ACM, 2007, pp. 24–32
[15] Berkeley UPC compiler, “upc.lbl.gov.”
[16] D. Bonachea, “GASNet specification v1.1,” Berkeley, CA USA, Tech. Rep., 2002.
[17] The GWU Unified Testing Suite – GUTS “threads.seas.gwu.edu/sites/guts.”
[18] D. Baily, E. Barscz, J. Barton, D. Browning, R. Carter L. Dagum, R. Fatoohi, S. Fineberg, et al., “The NAS parallel benchmarks”, NAS technical report rnr-94-007,”Moffett Field CA, USA, Tech. Rep., 1994.
[19] D. Bailey, T. Harris,W. Saphir, R. van der Wijngaart, A.Woo and M. Yarrow, “The NAS parallel benchmarks 2.0”, NAS technical report nas-95-020,” Moffett Field, CA, USA, Tech Rep., 1995.
[20] T. El-Ghazawi and F. Cantonnet, “UPC performance and potential: A NPB experimental study,” Nov. 2002, pp. 17–17.

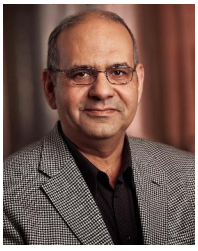
BIOGRAPHY

Olivier SERRES obtained the computer science engineering degree from the University of Technology of Belfort-Montbeliard. He is a doctoral student at the School of Engineering and Applied Science of the George Washington University (GWU). He is a research assistant at the High Performance Computing Laboratory of the GWU.



Ahmad Anbar graduated from the Faculty of Computer and Information Sciences (FCIS), Ain Shams University, Egypt in year 2000. He worked in teaching in the university since then. He also worked as a information analyst in Electronic Data Systems (EDS), Cairo branch for three years. Ahmad got his masters degree from Ain Shams University in 2006. His masters was about resources management in Grid environments. Since Fall 2008, Ahmad started his PhD in The George Washington University. He joined the High Performance Computing Lab (HPCL) as a research assistant. His main research in HPCL is targeting the support of UPC on many-core architectures.

Saumil Merchant received the B.E. degree in Electronics from Mumbai University, India in 1999, and the M.S. and PhD. degrees in Computer Engineering from University of Tennessee, Knoxville in 2003 and 2007 respectively. He is currently a research scientist in the department of Electrical and Computer Engineering at George Washington University. His research interests include reconfigurable computing, high-performance computing, embedded computing, and machine intelligence. He is a member of IEEE and ACM.



Tarek El-Ghazawi is a Professor in the Department of Electrical and Computer Engineering at The George Washington University, where he leads the university-wide Strategic Program in High-Performance Computing. He is the founding director of The GW Institute for Massively Parallel Applications and Computing Technologies (IMPACT) and a founding Co-Director of the NSF Industry/University Center for High-Performance Reconfigurable Computing (CHREC). El-Ghazawi’s research interests include high-

performance computing, computer architectures, reconfigurable, embedded computing and computer vision. He is one of the principal co-authors of the UPC parallel programming language and the first author of the UPC book from John Wiley and Sons. He has received his Ph.D. degree in Electrical and Computer Engineering from New Mexico State University in 1988. El-Ghazawi has published about 200 refereed research publications in this area. Dr. El-Ghazawi has served in many editorial roles and is currently an Associate Editor for the IEEE Transactions on Computers. He has chaired and co-chaired many international conferences and symposia including the 2009 Conference on Partitioned Global Address Space (PGAS) Programming Models and Languages (PGAS2009), The 10th IEEE International Conference on Scalable Computing and Communications (ScalCom-10), 2010, and the 9th ACS/IEEE Conference on Computer Systems and Applications, AICCSA2011. Dr. El-Ghazawi's research has been frequently supported by Federal agencies and industry. He serves or has served on many advisory boards including the Science Advisory Panel of the Arctic Region Supercomputing Center. Professor El-Ghazawi was elected to a Fellow of the IEEE with the citation "for contributions to reconfigurable computing and parallel programming".