# Efficient and Predictable Group Communication Messaging over Multi-core NoCs *

Karthik Yagna, Onkar Patil, Frank Mueller
North Carolina State University

## ABSTRACT

Massive multicore processors with network-on-chip (NoC) architectures are becoming common. Some such architectures provide native core-to-core communication via message passing, but are subject to NoC contention and execution time variance.

This work develops efficient and predictable group communication / collectives using message passing specifically designed for large core counts in 2D mesh NoC architectures. Collectives are implemented in such a way that they reduce NoC contention. Experimental results on a single-die 64 core hardware platform show significantly reduce communication times by up to 95% for single packet messages and up to 98% for longer messages with superior performance for mostly all (but sometimes only smaller) message sizes. In addition, our communication primitives have significantly lower variance than prior approaches, thereby providing more balanced parallel execution progress.

## 1. INTRODUCTION

Massive multicores with NoC architectures feature mesh-based NoCs for scalability. This allows a large number of cooperating tasks to be scheduled together. Tasks can employ group communication via messages over the NoC but may experience link contention and flow control become potential bottlenecks.

Poor group communication implementations can result in increased and highly variant latency due to NoC contention resulting in loss of predictability and imbalance in execution progress across cores. When multiple pairs of cores communicate, they may experience contention due to wormhole routing: After opening a source-destination path along a route, any other communication trying to use links on this path remains blocked until the former connection is closed. Such situations can be avoided using intelligent scheduling of each round of message exchanges.

Furthermore, NoC architectures provide multiple message queues and networks [2, 3, 28, 1]. On the TilePro64 [3], the User Dynamic Network (UDN) uses dynamic routing to forward messages from a source core to a destination core on Manhattan path following first the X- and then the Y-direction (X/Y-dimension ordered routing). The Static Network (SN) uses statically configured routes to forward packets received on each link. SN is faster than UDN in terms of packet forwarding speed (1 vs. 2 cycles), but is difficult to program and has route setup overhead.

This work contributes the design and implementation of group communication for large core counts utilizing 2D mesh NoC architectures. We employ efficient algorithms to reduce communication latency and exploit advanced NoC hardware features to provide better performance. We also ensure that communication uses contention-free paths and that no deadlock may occur. Deadlocks may occur due to head-of-line and path-based blocking, they are avoided by credit-based backpressure monitoring [34, 33] or by ensuring absence of link contention (this work, for collectives, only). We have implemented five commonly used group communication primitives [9].

Our Barrier, Broadcast and Reduce use a communication tree in which the cores are arranged as nodes and share a parent-child relationship. The communication tree is used to send messages to/from the root. The Barrier and Reduce implementations utilize the UDN, whereas Broadcast uses the SN. Our implementation of Alltoall exploits simple pattern-based communication, common in MPI [9] runtime system implementations, to send messages concurrently, yet without contention, to reduce communication latency. This neither requires dynamic computation of a routing schedule nor incurs scheduling overhead or memoization of large routing tables.

Experimental results on the TilePro hardware platform show that our implementation has lower latencies and less timing variability (lower variance) than prior work. We compared the performance of our implementation in micro-benchmarks against OperaMPI [13], a reference MPI implementation for the Tilera platform. Performance improvements of up to 95% are observed in communication for single packet messages with significantly higher timing predictability (lower variance), which supports more balanced execution progress for high-performance computing (HPC).

## 2. DESIGN AND IMPLEMENTATION

Our work assumes a generic, generalized 2D mesh NoC switching architecture similar to existing fabrics with high core counts [2, 3, 28, 15]. Each core is composed of a compute core, network switch, and local caches.

**NoC Message Layer:** Our implementation provides an MPI-style message passing interface for NoCs for basic point-to-point and group communication. The NoC message layer implementation optionally provides flow control support. In our design, we turn off flow control when not required by program logic to further improve performance.

**Group Communication Primitives:** The key ideas behind our design of group communication primitives are to (1) reduce contention in the NoC; (2) exploit pattern-based communication to exchange messages concurrently; (3) reduce the number of messages by aggregation; and (4) leverage hardware features to improve performance. Due to these objectives, it is not feasible to simply resort to binomial trees for most collectives or other algorithms such as recursive doubling for allreduce since these algorithms are contention agnostic and will result in reduced performance over contention-sensitive NoCs.

We implemented the group communication on the Tilera TilePro64 and ported it to the Intel SCC [15] to demonstrate that our imple-

*This work was supported in part by NSF grants 0905181 and 1239246.

mentation is generic and can be extended to any 2D mesh NoC architectures.

**Alltoall and Alltoallv:** Alltoall/Alltoallv employ pattern-based communication, which allows several sets of tasks to exchange messages concurrently without contention as many exchanges split into multiple rounds.

The rounds are comprised of (1) direct (2) left and (3) right rounds. The direct round is further split into two subrounds. In subrounds, each task sends messages only along a straight path to its partner task. Tasks exchange messages along the X direction in direct subround 1 and along the Y direction in direct subround 2. In left rounds, each task sends messages along the X direction followed by the Y direction such that their path follows a counter-clockwise direction. In right rounds, each task sends messages along the X direction followed by the Y direction such that their paths follow a clockwise direction. These cases are depicted in Figure 1. The X/Y dimension routing ensures that these directions are maintained consistently.

The next rounds are expanded on horizontal paths to 2 hops, then for 2-hop vertical paths, followed by 3-hop rounds and so on. The implementation details are omitted due to space (see [30]).

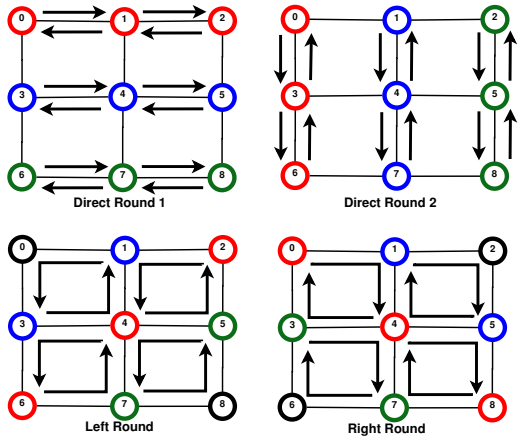


Figure 1: Alltoall Rounds

**Barriers:** We utilize modified 3-ary tree-based barriers that distribute the work evenly among nodes to minimize cycle differences upon barrier completion. In a 2D mesh, nodes have a most 4 neighbors so that in a barrier tree, any interior node receives a message from one neighbor and relays this message to 3 others. This provides maximal link coverage with minimal tree height (which is optimal). Hence, the tree is 3-ary on the interior, 4-ary for the root (to be precise) and of lower degree (2/1/0) for nodes close to the leaves and leaves themselves. The root of this tree is placed in the center of the NoCMsg grid to minimize latency (hops). The tree is constructed as part of the initialization process. Children notifying their parents when they have entered the barrier, up to the root. Once the root has received notifications from all children, it broadcasts a notification back down the tree by replying to its children and returns from the barrier call, as do the children. UDN is used to send/receive synchronization packets and their replies.

**Broadcast:** Our Broadcast uses the SN of the TilePro64. The SN is more intricate to program and suffers from route setup overhead. However, message forwarding incurs zero overhead (due to a static route configuration). Since broadcast has a single sender and multiple receivers, the number of route configurations is low. This was the motivation behind using SN for the broadcast implementation.

We designed a tree-based algorithm rooted at the task performing the broadcast. Each task determines the root's row and column and configures the SN route. The route setup in the root is such that the message from the core is sent on its available links. All the tasks in the same column as the root have their route configured such that they receive from the root along the Y direction and send the message along other available links. Tasks in other columns receive along one X direction and send the message along the other X link.

The static route of each task is configured inside the Broadcast call such that the message from the root flows to each leaf task. Our current implementation requires only a single route configuration per task and is contention-free.

**Reduce and AllReduce:** Our Reduce is similar to the barrier. The reduction operation is performed along the tree. Each child task sends its partial result upward toward the root. The root reduces the partial results to obtain the final result. The construction of the reduction tree is different from that of the Barrier. The reduction tree maps to a NoC grid such that the root task becomes the root of the tree. The tasks along its row become first-level children. The tasks in each column become second-level children to the first-level ones. Via recursive refinement, the algorithm extends to larger meshes, where more levels would be employed.

AllReduce is an extension of Reduce. It is implemented by performing a Reduce relative to rank 0, followed by a broadcast from rank 0 to all other tasks in the group.

# 3. FRAMEWORK

Experiments were conducted on a 700MHz 64-core Tilera TilePro processor (TilePro 64) and the Intel SCC. Our implementation is written in C and the programs were compiled with Tilera's MDE 3.03 tool chain/Intel's ICC at the O3 optimization level with respective C/C++/Fortran compilers. Some cores are used for specific purposes and are not available to run user programs restricting the maximum grid size to 7x7 on Tilera. In our experiments, grid sizes ranging from 2x2 to 7x7 were utilized allowing for a total of six configurations (omitting rectangular, non-square configurations, which are also valid).

OperaMPI implements the MPI 1.2 standard [10] for C. It is layered over Tilera's iLib, an inter-tile communication library that utilizes the UDN NoC network. The iLib library is vendor-supplied software and supports point-to-point messages. The iLib utilizes interrupt-based virtual channels and complex packet encodings to synchronize senders and receivers to set up such point-to-point communication. However, iLib only supports a limited number of collective operations, namely broadcast and barrier. Hence, OperaMPI creates virtual overlays (e.g., binomial trees for reductions, recursive doubling for allreduce) to implement more complex MPI collectives, which result in NoC contention in contrast to our approach. OperaMPI has been evaluated against the IBM, Intel, MPICH and SPEC MPI test suites. Some of these results and implementation details (such as tree reduce/alltoall) are discussed in [13], which provides a fair foundation for comparison to OperaMPI. Furthermore, OperaMPI is the only working MPI implementation on Tilera. Other ports (MPICH or Open MPI) are not only beyond the scope of this work but would be subject to the same NoC design/implementation choices we made.

Results of NoCMsg, our scalable and predictable NoC messaging framework, and the third-party OperaMPI and RCKMPI [29, 8] were compared for Tilera and SCC, respectively. NoCMsg traffic is routed over the UDN/SN with 1-2 cycles and 1 cycle per hop, respectively, using polling instead of relying on iLib/OperaMPI interrupts with messages packaged in 128 byte flits.

# 4. EXPERIMENTAL RESULTS

We experimented with our group communication layer, embedded in NoCMsg, on the Tilera TilePro64 while Intel SCC [15] re-

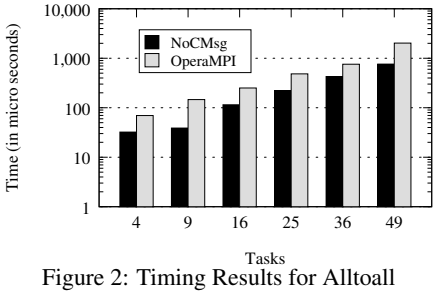

Figure 2: Timing Results for Alltoall

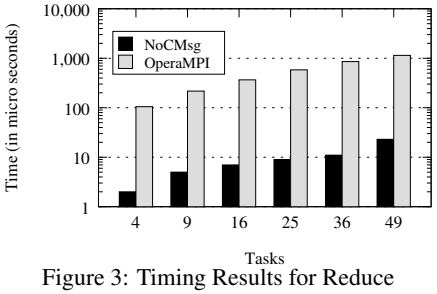

Figure 3: Timing Results for Reduce

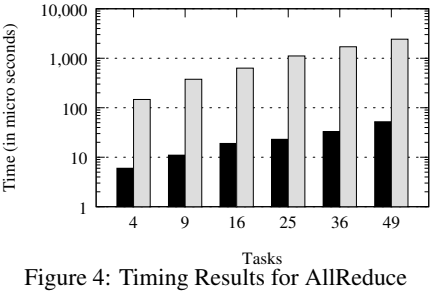

Figure 4: Timing Results for AllReduce

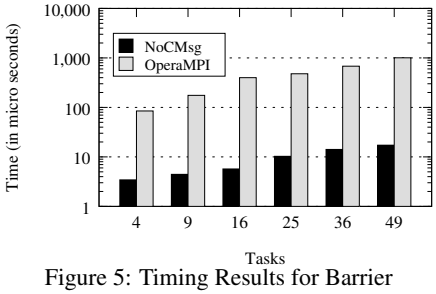

Figure 5: Timing Results for Barrier

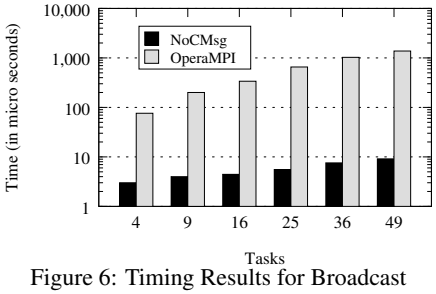

Figure 6: Timing Results for Broadcast

sults are omitted due to space (see [19]).

## 4.1 Single packet messages

The benchmark timing results for single packet messages are depicted in Figures 2-6 for 5 collectives. Time on the y-axis is plotted logarithmically in microseconds for averaged benchmark runs over different number of tasks (equal to cores) in the range from 4 to 49. Tables 1 and 2 depict execution time variances for each micro-benchmark for varying number of tasks for NocMsg and OperaMPI, respectively.

Table 1: NoCMsg Execution Time Variance $[\mu s^2]$

| Num tasks | 4 | 9 | 16 | 25 | 36 | 49 |
|---|---|---|---|---|---|---|
| Alltoall | 0.7 | 0.4 | 0.7 | 5.6 | 1.3 | 1.6 |
| Barrier | 0.5 | 0.8 | 0.4 | 1.6 | 1.1 | 5.6 |
| Broadcast | 0 | 0 | 0.2 | 0.24 | 0.53 | 0.12 |
| Reduce | 0.2 | 0.12 | 1.1 | 1.06 | 3.96 | 7.27 |
| AllReduce | 1.34 | 2.29 | 2.49 | 26.77 | 31.55 | 50.82 |

Table 2: OperaMPI Execution Time Variance $[\mu s^2]$

| Num tasks | 4 | 9 | 16 | 25 | 36 | 49 |
|---|---|---|---|---|---|---|
| Alltoall | 2.81 | 983.9 | 18.2 | 2276.8 | 133329.8 | 622903 |
| Barrier | 750.2 | 302.9 | 29384.5 | 1838.2 | 2910.7 | 32117 |
| Broadcast | 7.3 | 56.9 | 259.2 | 4540.8 | 3003.7 | 3869 |
| Reduce | 545.26 | 686.2 | 21.39 | 2007.06 | 9979.96 | 3430.69 |
| AllReduce | 11.14 | 50.98 | 49.36 | 3839.44 | 5536.16 | 7517.2 |

We observe that as the number of tasks increases, the execution time of group communication increases. In case of Opera, the increase in runtime is significant for larger number of tasks. In comparison, our NoCMsg implementation is highly efficient, and increases in runtime are gradual. For Alltoall, our pattern-based approach effectively eliminates network contention resulting in a reduction of execution time by about 62% for a 7x7 grid size with a variance of 0.4 to 5.6 depending on the numbers of cores used. This variance is several orders of magnitude lower than that of the OperaMPI particularly for larger number of cores.

Barrier and Broadcast are our most efficient collectives with up to 98% reduction in execution time, i.e., Opera takes up to two orders of magnitude longer due to contention. Broadcast uses the SN with a single route setup (to configure the communication tree) and minimal routing overhead. The SN is typically faster than the UDN, which makes Broadcast our most efficient and predictable collective in comparison. (Over UDN, broadcast would be close to the time of a barrier.) Execution time increases only by a factor of 3.5 as the grid size is gradually changed from 2x2 to 7x7 with a variance of less than 0.6 for all cases.

Our implementations of Reduce and Allreduce have 97% and 98% lower execution time, respectively, than OperaMPI for all tested grid sizes, i.e., OperaMPI is up to nearly two orders of magnitude slower. However, these collectives have a larger variance than others due to the two-step reduction employed by Reduce operations.

## 4.2 NAS Parallel Benchmark

We used NPB version 3.3 to evaluate our implementation. NPB by default uses strong scaling, where input sizes stay fixed for different numbers of cores. We used strong scaling due to input constraints for MG (input class A, to mostly remain on chip in L2 cache) and our own weak scaling inputs for all other benchmarks [11]. Weak scaling ensures that the computational work per core remains the same as the number of cores cooperating in a parallel application is increased. We choose inputs to remain resident in private L2 so that results measure the NoC properties rather than being dominated by off-chip memory traffic.

For MG with strong scaling (Figure 7), NoCMsg is faster than OperaMPI for all grid sizes. Strong input scaling causes the total time to reduce as the number of tasks increases. For small task sizes, NoCMsg is much faster than OperaMPI. As the number of tasks increases, the time difference decreases. This is because MG is memory intensive with limited inter-process communication, and for large grid sizes the performance improvement due to efficient communication declines.

For IS and CG (Figures 8+9), the execution time of NoCMsg is lower than that of OperaMPI. Performance benefits include both collectives and point-to-point improvements with improvements due to collectives (this paper) and point-to-point messages (for the latter, see [34, 33]). The difference in execution time increases with as the number of tasks grows since inter-process communication dominates. For FT (Figure 10), small differences in execution time between NoCMsg and OperaMPI are due to larger messages sizes.

For LU (Figure 11), OperaMPI is faster than NoCMsg for small numbers of tasks when computation dominates total execution time. As the number of tasks increases, the inter-task communication starts to dominate the total execution time. The execution time of

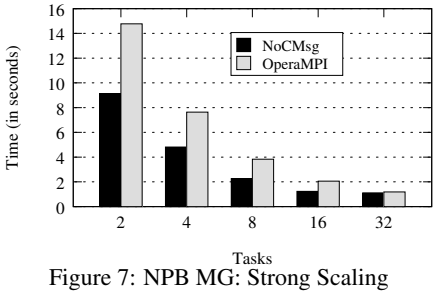

Figure 7: NPB MG: Strong Scaling

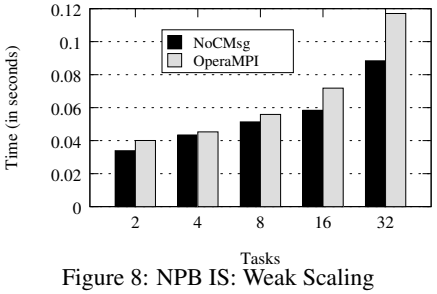

Figure 8: NPB IS: Weak Scaling

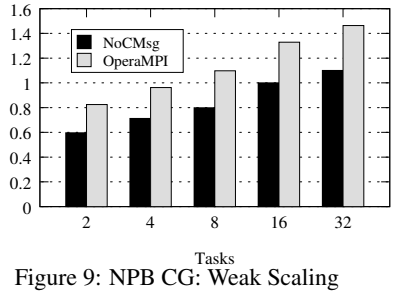

Figure 9: NPB CG: Weak Scaling

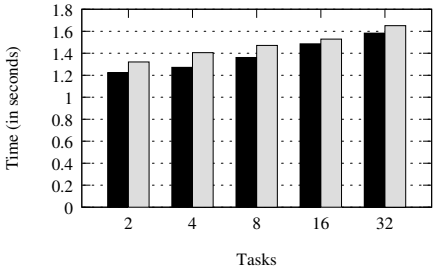

Figure 10: NPB FT: Weak Scaling

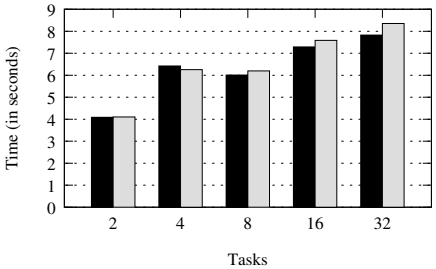

Figure 11: NPB LU: Weak Scaling

NoCMsg grows slower than that of OperaMPI, indicating that for larger numbers of tasks NoCMsg provides better performance.

# 5. RELATED WORK

Communication patterns and communication trees as a means to implement collective operations have been well studied [16]. Barnett's broadcast [5] is loosely based on spanning binomial trees. Yang's tree-based multicast [31] constructs a quad-branch multicast (QBM) tree, a logic tree rooted at the source node of a multicast and has four subtrees. Our implementation of Broadcast uses a communication tree in a contention-free manner. But unlike QBM, our approach does not require special registers/buffers in the routers, support of double-X/Y routing and changes to message headers. QBM also cannot handle alltoallv. Topology-aware collectives [21] reduce but cannot eliminate congestion for HPC interconnect whereas our work eliminates any contention.

Several approaches apply graph theoretic concepts to build efficient trees, e.g., a multi-level broadcast tree using extended dominator nodes (EDN) [27] but requires an expensive all-port instead of the current single-port NoC communication architectures.

Barriers were implemented in hardware for IBM's BlueGene [20]. Router-based barriers [14, 18] require long headers to carry the information of multiple destinations and incur additional processing at each node. Tree-based barriers [32] partition the 2D mesh into four overlapping quadrants using the chosen root node as the origin whereas Barrier Tree for Meshes (BTM) utilizes a 4-ary synchronization tree constructed in a recursive manner [17]. Our implementation of barrier also uses a tree rooted at a chosen root node, but, unlike the others, neither requires a 2D division into submeshes nor relies on special registers for tree construction.

Our implementation of alltoall exploits pattern-based communication to concurrently exchange messages between partners. On the surface, this approach shares design strategies with Thakur's direct algorithm [26], which assigns mesh nodes ordinal numbers 0..N-1 in a row-major fashion. During step $k$, for $k = 1..N-1$, the node with ordinal number $i$ sends a message to the node whose ordinal number is an exclusive or (XOR) of $i$ and $k$. This results in a communication pattern similar to ours under dimension order routing. Unlike our approach, their direct algorithm suffers from link contention, which is also the case for BG/L's random lists [4]. Others split tasks into disjoint communication groups [6, 25, 23, 24]. Our implementation also uses a bottom-up approach, but neither requires a grid division into smaller submeshes nor results in contention.

More recent approaches focus on building static schedules for alltoall communication [7]. Some approaches perform path selection, core mapping and time-slot allocation intelligently to resolve conflicts on shared networks [12]. Others exploit time-division multiplexing on NoCs to solve the slot and path selection problem and this avoid contention [22]. Unlike these approaches, our implementation neither requires dynamic route calculations nor offline pre-calculations nor storage of large routing tables. Our tree-based implementations rely on the relative position of nodes to the root and take advantage of 2D mesh topology to map a tree in a contention-free manner, which is novel. This keeps our implementation simple, generic and scalable with minimum overhead.

# 6. CONCLUSION

We designed a set of efficient and predictable group communication primitives using message passing utilizing NoC architectures. The primitives employ highly efficient algorithms to provide contention-free communication and utilize advanced NoC hardware features. These primitives improve performance and reduce imbalance for HPC applications.

Our implementation of the most commonly used collectives reduces the communication time over a reference MPI implementation on TilePro64 by up to 95% for single packet messages and up to 98% for larger messages (see [30]). NocMsg has superior performance over OperaMPI irrespective of message size for all but one collective: For Alltoall, NocMsg performs better for message sizes up to 256 Bytes while OperaMPI performs better for larger messages. The evaluation of NPB codes shows that NoCMsg outperforms OperaMPI for actual workloads. NocMsg thus nicely complements prior work that is efficient at larger (yet less common) message sizes for this case. Additionally, the variance of execution times for our implementation is several orders of magnitude lower than that of the reference MPI implementation, making our implementation ideal for balanced high-end as well as embedded/real-time applications. And instead of assuming ideal NoC symmetry with wrap-around links on the 2D boundaries, our work addresses realistic 2D meshes without wrap-around, such as present in contemporary NoC hardware designs.

# 7. REFERENCES


[1] Adapteva processor family. `www.adapteva.com/products/silicon-devices/e16g301/`.

[2] Tera-scale research prototype: Connecting 80 simple cores on a single test chip. ftp://download.intel.com/research/platform/terascale/terascaleresearchprototypebackgrounder.pdf.

[3] Tilera processor family. `www.tilera.com/products/processors.php`.

[4] George Almási, Philip Heidelberger, Charles J. Archer, Xavier Martorell, C. Chris Erway, José E. Moreira, B. Steinmacher-Burow, and Yili Zheng. Optimization of mpi collective communication on bluegene/l systems. In *International Conference on Supercomputing*, pages 253–262, 2005.

[5] Michael Barnett, David G. Payne, and Robert A. van de Geijn. Optimal broadcasting in mesh-connected architectures. Technical report, University of Texas at Austin, Austin, TX, USA, 1991.

[6] S.H. Bokhari and H. Berryman. Complete exchange on a circuit switched mesh. In *Scalable High Performance Computing Conference, 1992. SHPCC-92, Proceedings.*, pages 300–306, 1992.

[7] Florian Brandner and Martin Schoeberl. Static routing in symmetric real-time network-on-chips. In *International Conference on Real-Time and Network Systems*, pages 61–70, 2012.

[8] IsaiasA. Compres Urena, Michael Riepen, and Michael Konow. Rckmpi âĂŞ lightweight mpi implementation for intelâĂŹs single-chip cloud computer (scc). *Recent Advances in the Message Passing Interface*, pages 208–217, 2011.

[9] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European PVM/MPI Users' Group Meeting*, pages 97–104, September 2004.

[10] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.

[11] John L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, May 1988.

[12] Andreas Hansson, Kees Goossens, and Andrei Rădulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '05, pages 75–80, New York, NY, USA, 2005. ACM.

[13] M. Kang, E. Park, M. Cho, J. Suh, D.-I. Kang, and S. P. Crago. Mpi performance analysis and optimization on tile64/maestro. In *Workshop on Multi-core Processors for Space — Opportunities and Challenges*, July 2009.

[14] X. Lin, P. K. McKinley, and L. M. Ni. Deadlock-free multicast wormhole routing in 2-d mesh multicomputers. *IEEE Trans. Parallel Distrib. Syst.*, 5(8):793–804, August 1994.

[15] T.G. Mattson, R.F. van der Wijngaart, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe. The 48-core scc processor: the programmer's view. In *Supercomputing*, Nov 2010.

[16] Philip K. McKinley, Yih jia Tsai, and David F. Robinson. A survey of collective communication in wormhole-routed massively parallel computers. *IEEE COMPUTER*, 28:39–50, 1994.

[17] Sangman Moh, Chansu Yu, Ben Lee, Hee Young Youn, Dongsoo Han, and Dongman Lee. Four-ary tree-based barrier synchronization for 2d meshes without nonmember involvement. *IEEE Trans. Comput.*, 50(8):811–823, August 2001.

[18] D. K. Panda. Fast barrier synchronization in wormhole k-ary n-cube networks with multidestination worms. In *Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture*, HPCA '95, pages 200–, Washington, DC, USA, 1995. IEEE Computer Society.

[19] O. Patil. Effcient and lightweigth inter-process collective operations for massive multi-core architectures. Master's thesis, North Carolina State University, June 2014.

[20] Vara Ramakrishnan and Isaac D. Scherson. Efficient techniques for nested and disjoint barrier synchronization. *J. Parallel Distrib. Comput.*, 58(2):333–356, August 1999.

[21] Paul Sack and William Gropp. Faster topology-aware collective algorithms through non-minimal communication. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 45–54, 2012.

[22] Radu Stefan and Kees Goossens. An improved algorithm for slot selection in the ÃĘthereal network-on-chip. In *International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*, pages 7–10, 2011.

[23] Young-Joo Suh and Kang G. Shin. All-to-all personalized communication in multidimensional torus and mesh networks. *IEEE Trans. Parallel Distrib. Syst.*, 12(1):38–59, January 2001.

[24] Young-Joo Suh and Sudhakar Yalamanchili. All-to-all communication with minimum start-up costs in 2d/3d tori and meshes. *IEEE Trans. Parallel Distrib. Syst.*, 9(5):442–458, May 1998.

[25] N. S. Sundar, D. N. Jayasimha, D.K. Panda, and P. Sadayappan. Complete exchange in 2d meshes. In *Scalable High-Performance Computing Conference, 1994., Proceedings of the*, pages 406–413, 1994.

[26] Rajeev Thakur and Alok Choudhary. All-to-all communication on meshes with wormhole routing. In *In Proceedings of the 8 th International Parallel Processing Symposium*, pages 561–565, 1994.

[27] Yih-jia Tsai and Philip K. McKinley. Broadcast in all-port wormhole-routed 3d mesh networks using extended dominating sets. In *Proceedings of the 1994 International Conference on Parallel and Distributed Systems*, pages 120–127, Washington, DC, USA, 1994. IEEE Computer Society.

[28] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27:15–31, 2007.

[29] Rob Van Der Wijngaart and Tim Mattson. RCCE: A small library for many-core communication, 2010.

[30] K. Yagna. Collective communication for multi-core noc interconnects. Master's thesis, North Carolina State

University, May 2013.

[31] Jenq-Shyan Yang and Chung-Ta King. Efficient tree-based multicast in wormhole-routed 2d meshes. In *Proceedings of the 1997 International Symposium on Parallel Architectures, Algorithms and Networks*, ISPAN '97, pages 494–, Washington, DC, USA, 1997. IEEE Computer Society.

[32] Jenq-Shyan Yang and Chung-Ta King. Designing tree-based barrier synchronization on 2d mesh networks. *IEEE Trans. Parallel Distrib. Syst.*, 9(6):526–534, June 1998.

[33] C. Zimmer and F. Mueller. Nocmsg: Scalable message passing abstraction for network-on-chips. *ACM Transactions on Architecture and Code Optimization*, page (accepted), 2014.

[34] C. Zimmer and F. Mueller. Nocmsg: Scalable noc-based message passing. In *International Symposium on Cluster Computing and the Grid (CCGRID)*, page (accepted), 2014.