LA-UR-18-23228

MPI- RMA: The State of the Art in Open MPI

SEA workshop 2018, UCAR

Howard Pritchard
Nathan Hjelm
April 6, 2018

Los Alamos NATIONAL LABORATORY EST.1943

NNSA National Nuclear Security Administration

Acknowledgments – who's done most of the work covered today

- Nathan Hjelm (LANL)
- Matthew Dosanjh (SNL)
- Taylor Groves (LBNL)
- Ryan Grant (SNL)

More Acknowledgements

- This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.
- Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
- This work was performed in part at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract DE-FC02-06ER25750.

What we'll cover

- Motivators for enhancing MPI RMA performance
- Open MPI implementation of MPI RMA
- Recent improvements to Open MPI's RMA implementation
- Results using micro-benchmarks and an application (WOMBAT)
- What's next

ECP MPI Usage Survey

Survey Motivation

- **ECP represents a broad cross-section of computational science and engineering applications, numerical libraries, and other tools**
- ***Need* and *opportunity* to investigate how this community uses and plans to use MPI**
  - Ensure project plans align with community needs
  - Identify gaps both now and for apps at exascale
  - Explore this community's reactions to some of the questions facing the MPI Forum
  - Help shape new proposals to the Forum
- **Help us (OMPI-X project) identify potential partners for demonstration and validation**

MPI Usage Patterns

Feature	Current Usage	Exascale Usage
Point-to-point	**88%**	**80%**
MPI derived data types	23%	21%
Collectives	**80%**	**82%**
Neighbor collectives	14%	↔ 29%
Communicators and group mgmt	**61%**	**55%**
Process topologies	11%	21%
RMA	21%	↔ 43%
RMA shared windows	12%	↔ 20%
MPI I/O (called directly)	21%	20%
MPI I/O (via library)	27%	30%
MPI profiling interface	14%	16%
MPI tools interface	2%	9%

MPI Survey: MPI + Threads

- **Most apps plan to use multiple threads in an MPI process: 79% AD, 93% ST**
- **MPI calls from multi-threaded regions of code?**
  - Point-to-point 45%, RMA 29%, collectives 20%, not important 18%
- **General concerns about interference between MPI runtime and the threading model runtimes**
- **General concerns about performance of MPI_THREAD_MULTIPLE**

Other Motivators

Other Reasons to Optimize MPI RMA

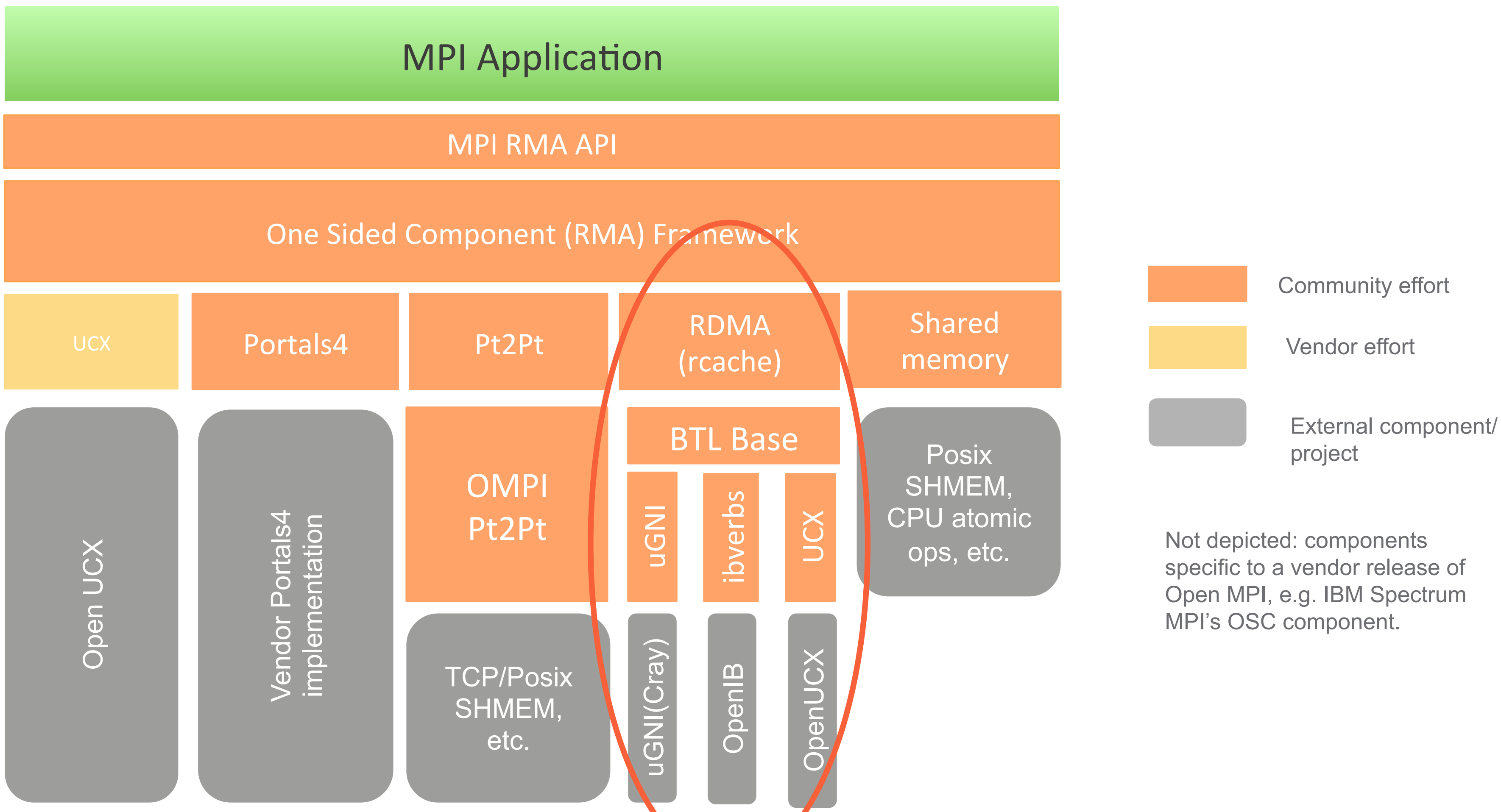
- **RDMA capable networks typically support one-sided remote memory access (put/get) better than send/recv style data exchange patterns:**
  - Generally easier to offload from host CPU
  - Tag matching hardware making progress but still more resource constrained than much simpler put/get path through the NIC
- **MPI RMA model decouples data movement from synchronization**
- **RMA imposes fewer requirements for message ordering in the network**
- **In theory, easier to realize good MPI RMA performance when used within multi-threaded regions of applications**
  - No tag matching sequential regions
  - No ordering of MPI RMA requests without explicit synchronization points

Some challenges

- **Inertia – slow adoption by applications due to sub-optimal performance of MPI RMA implementations**
- **MPI-2 had limitations with RMA epoch options, complicated memory consistency semantics, etc.**
- **Significant improvements with MPI-3 should help**
  - MPI_Win_lock_all
  - MPI_Win_flush
  - Request based RMA operations (MPI_Rget, etc.)
  - Shared memory windows

Open MPI RMA Implementation

Open MPI’s RMA Framework (OSC)



Not depicted: components specific to a vendor release of Open MPI, e.g. IBM Spectrum MPI’s OSC component.

OSC Components

- **Point to point component uses MPI send/recv operations to emulate MPI RMA one-sided operations. The original way Open MPI supported (prior to release 2.0.0) MPI RMA. Portable but not high performance.**
- **UCX component utilizes OpenUCX put/get/atomic support. Currently does not support thread-level concurrent access to network resources.**
- **Portals4 component utilizes Portals4 put/get/atomic support. Currently does not support thread-level concurrent access to network resources.**
- **Shared memory – special component for single node window operations.**
- **RDMA component utilizes underlying Open MPI RMA capable Byte Transport Layer (BTL) components to support MPI RMA.**

Recent RMA enhancements – BTL changes

- Change interfaces to BTL get/put methods to improve small RMA request throughput
- Add support for 32/64 bit atomic memory operations – compare and swap, fetch and add (integer and floating point)
- When MPI is initialized with MPI_THREAD_MULTIPLE, enable use of multiple network endpoints per MPI process – assign application threads round-robin.   Reduces contention for locks around network API interface calls.
- Still route send/recv traffic through a single network endpoint to preserve message ordering

Recent RMA enhancements – Rcache changes

- **Previously registration cache (rcache) used a single lock on the rcache to lookup, or insert, delete an entry**
- **Was based on a red-black + doubly linked list**
- **Replaced with interval tree, relativistic ordering, there is now only a write lock to do node insertion, rotation, or deletion, so typical read-only lookup's are fast**
- **Based on work by *P. W. Howard and J. Walpole(2014), Relativistic red-black trees, Concurrency Computat.: Pract. Exper., 26, pages 2684–2712.***

Recent RMA enhancements – RDMA component (1)

- **Scalable memory scheme for storing window base addresses and optional memory keys required by some RDMA networks**
  - Use a block of shared memory memory per node to cache local window base address/memory key information
  - Cache on a per node basis the address/memory key information of the above cache for other nodes in the job – O(N) N == #nodes in the job
  - As required by RMA operations, a process retrieves the required base address/memory key for a target process by first looking up remote nodes cache info, then fetch the right base address/memory from the remote cache
  - Scheme targets nodes with high core counts/CPU

Recent RMA enhancements – RDMA component (2)

- **Lock Scaling Improvements for MPI_Win_lock_all (two approaches)**
  - On demand locking
    - Single lock per window/per process. Uses atomic fetch_and_add based locking scheme. Best suited for applications that either do not use MPI_Win_lock_all or else make extensive use of MPI_Win_lock with exclusive attribute.
  - Two-level locking
    - Enables MPI_Win_lock_all without a lock operation per target MPI process
    - Single 64 bit global counter held by relative rank ‘0’ of the MPI window group + a per process lock (similar to on demand lock above)
    - 32 bits of counter count # lock all shared operations in progress, other 32 bits #lock exclusive operations in progress.
    - If a process wants to lock_all, okay if #lock all shared ops > 0, not okay if #lock exclusive ops is > 0.
    - If a process wants to lock exclusive, fail if #lock all shared ops > 0, okay if #lock exlcusive ops > 0, but must also try to acquire lock at target process
    - Best for case where MPI_Win_lock_all used frequently, lock exclusive rarely used

MPI-RMA Multi-threaded benchmark results (RMA-MT)

RMA-MT Multi-threaded benchmarks

- **Written because there were no existing benchmarks for measuring MPI RMA multi-threaded performance**
- **Tests measure**
  - Latency
  - Bandwidth (uni and bi-directional)
  - Message rate (multiple pairs of processes)
    - Single directional bandwidth
    - Halo exchange style pattern
  - Tests 4 RMA synchronization methods (fence, PSCW, lock/unlock, and lock_all/unlock_all)
- **3 mini-apps (HPCCG, MiniFE, MiniMD)**
- **Working through Sandia Labs/DOE legal to get open-sourced**

RMA-MT Bandwidth Put/Get Benchmark – Experimental setup

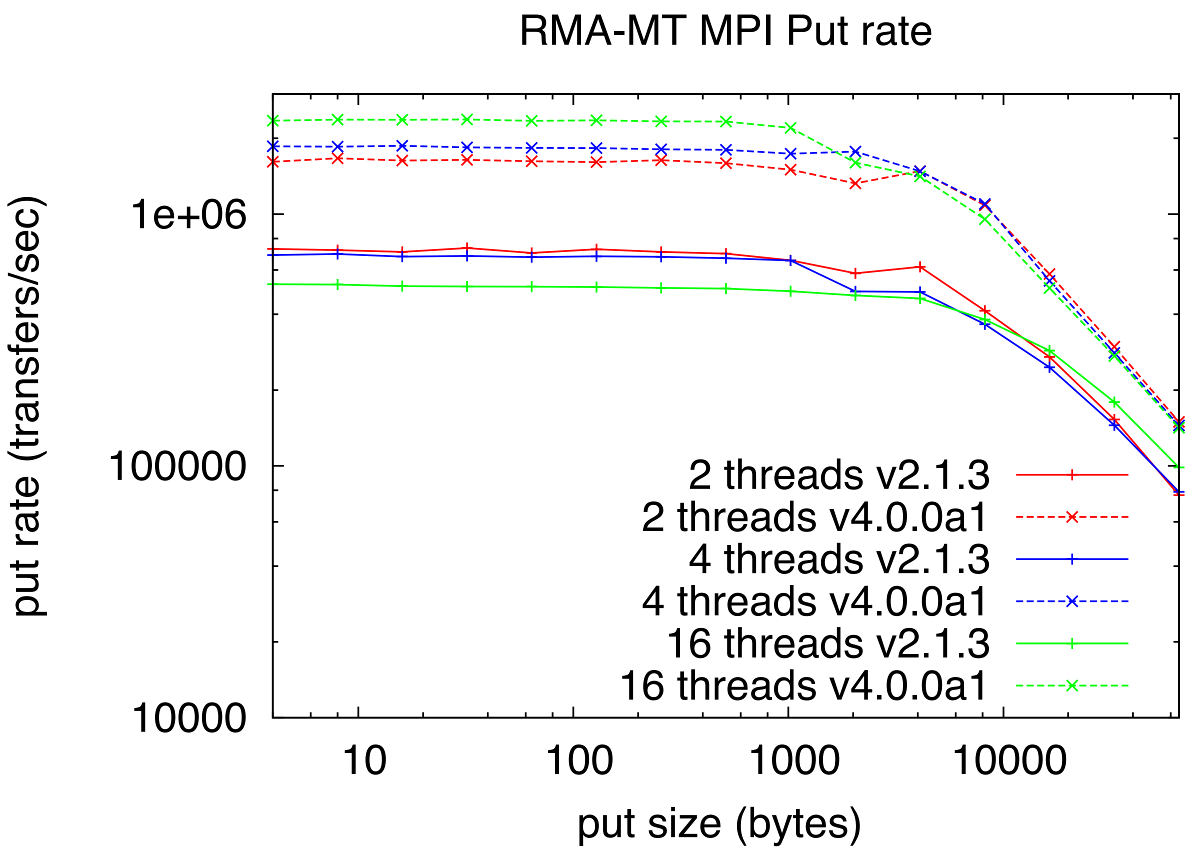
- Cray XC – Haswell processors (2.3 GHz, 32 cores/node)
- OS CLE 6.0UP05
- GNU C 7.2.0
- Compare Open MPI 4.0.0 pre-release with 2.1.3
- Cray Aries Network – so we're using the uGNI BTL within Open MPI
- Use Aries FMA (PIO based) network access method for transfers less than 2048 bytes. Larger transfers use Aries RDMA engine. The RDMA engine has 4 virtual channels.

RMA-MT Put/Get Bandwidth Benchmark

- **MPI Processes on paired nodes create a MPI window**
- **MPI Process on first node in pair spawns N child threads to do RMA operation**
- **Child threads and main thread sync**
- **Loop over Put sizes**
  - Warm up loop
  - Sync with other child threads
  - Start timer
  - Open exposure epoch (MPI_Win_lock shared for these results)
  - Loop over MPI_Put or MPI_Get operations (1000 for these results)
  - Call MPI_Win_flush
  - Stop timer
  - Sync with other child threads
  - Continue to next transfer size
- **Sync with main thread (pthread join)**
- **Ibarrier with MPI process on paired node**

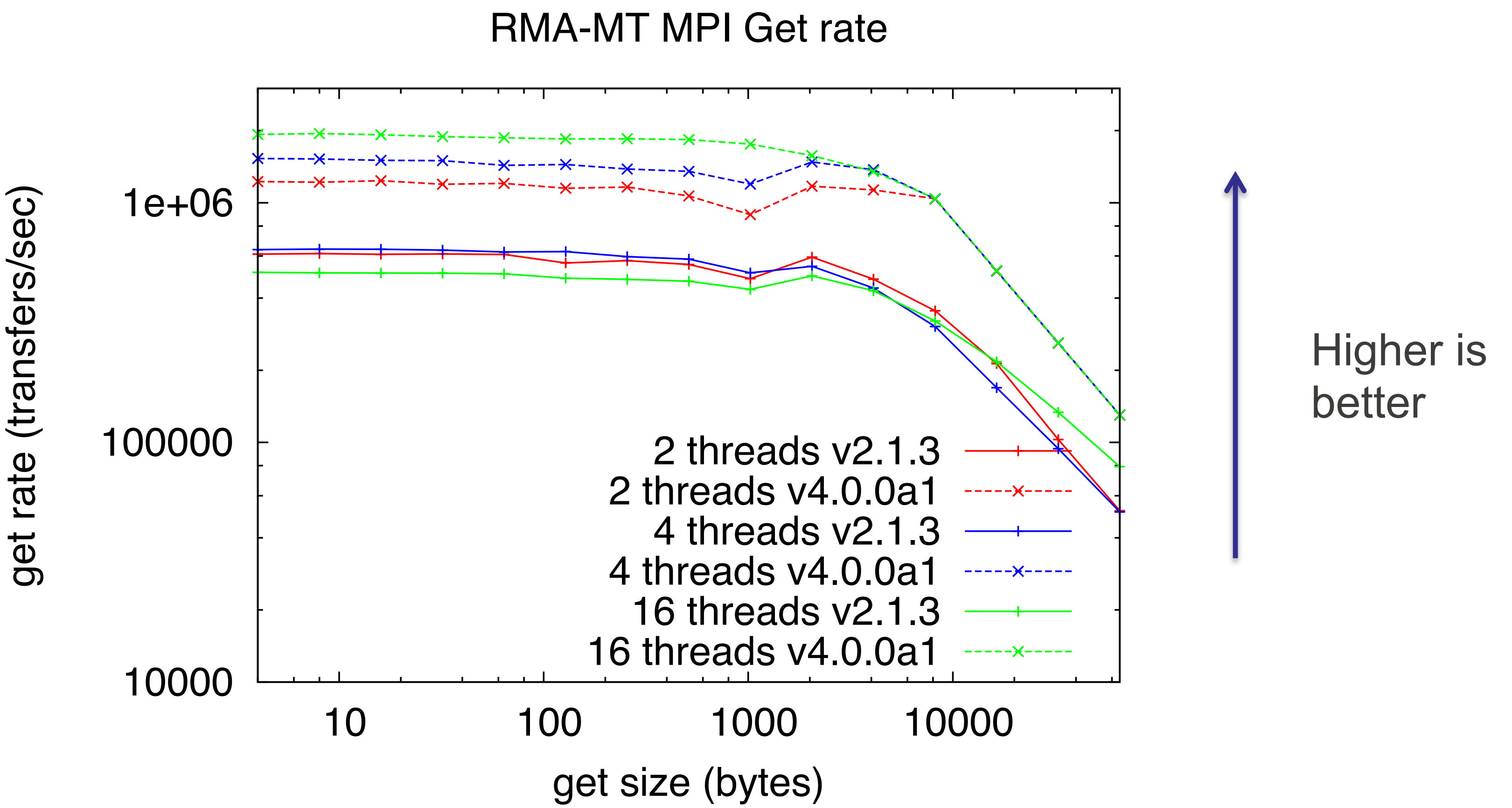
Note threads share same window

RMA-MT Put Bandwidth



Higher is better

RMA-MT Get Bandwidth

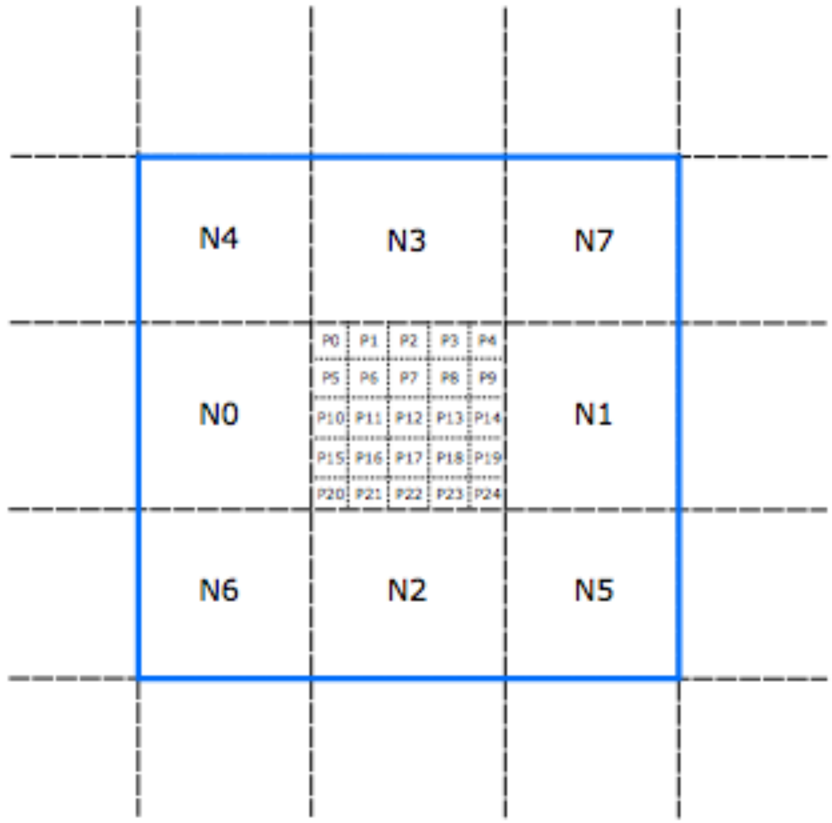


Wombat Application

Wombat

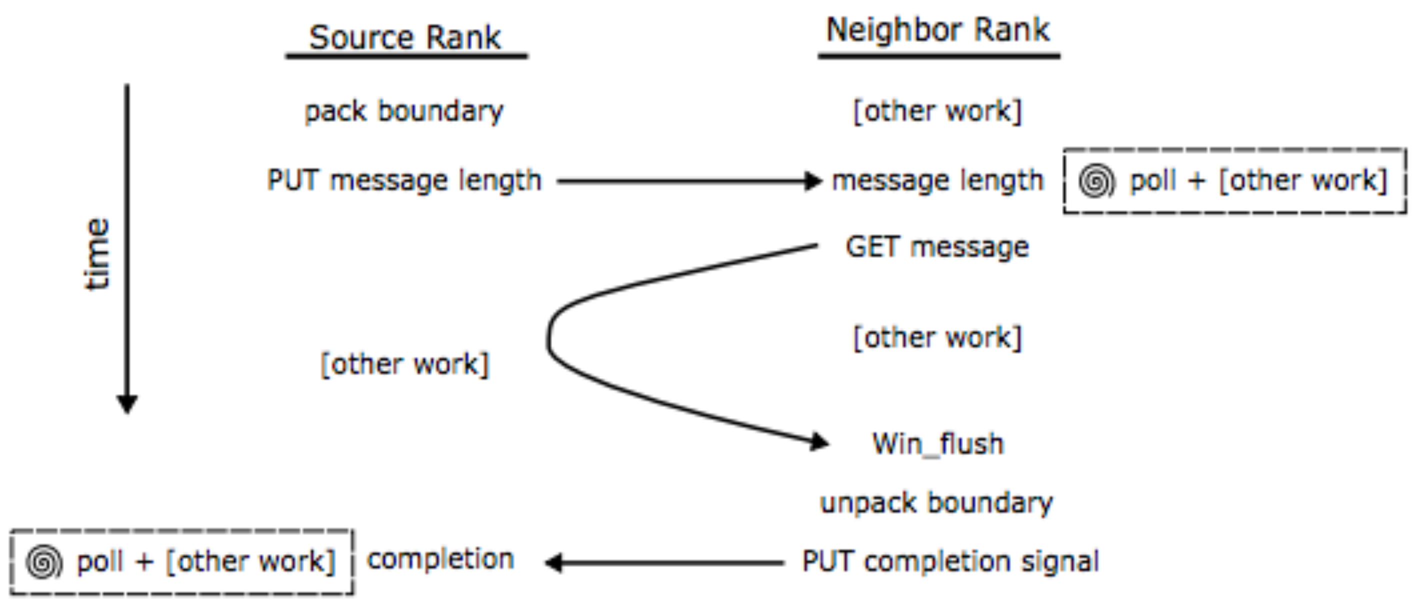
- **Astrophysics magneto-hydrodynamics code**
- **Collaborative effort between Cray Inc. and Univ. Minnesota**
- **Uses MPI/OpenMP targeting multi/many core processors**
  - Single large OpenMP region, try to limit OpenMP synchronization points
  - Uses MPI_THREAD_MULTIPLE
  - Use MPI-RMA to exchange patch edge data (passive synchronization) – within OpenMP region
- **Fine grain decomposition to better overlap communication with computation and communication with other communication – avoid BSP model**

Wombat – 2d patch exchange

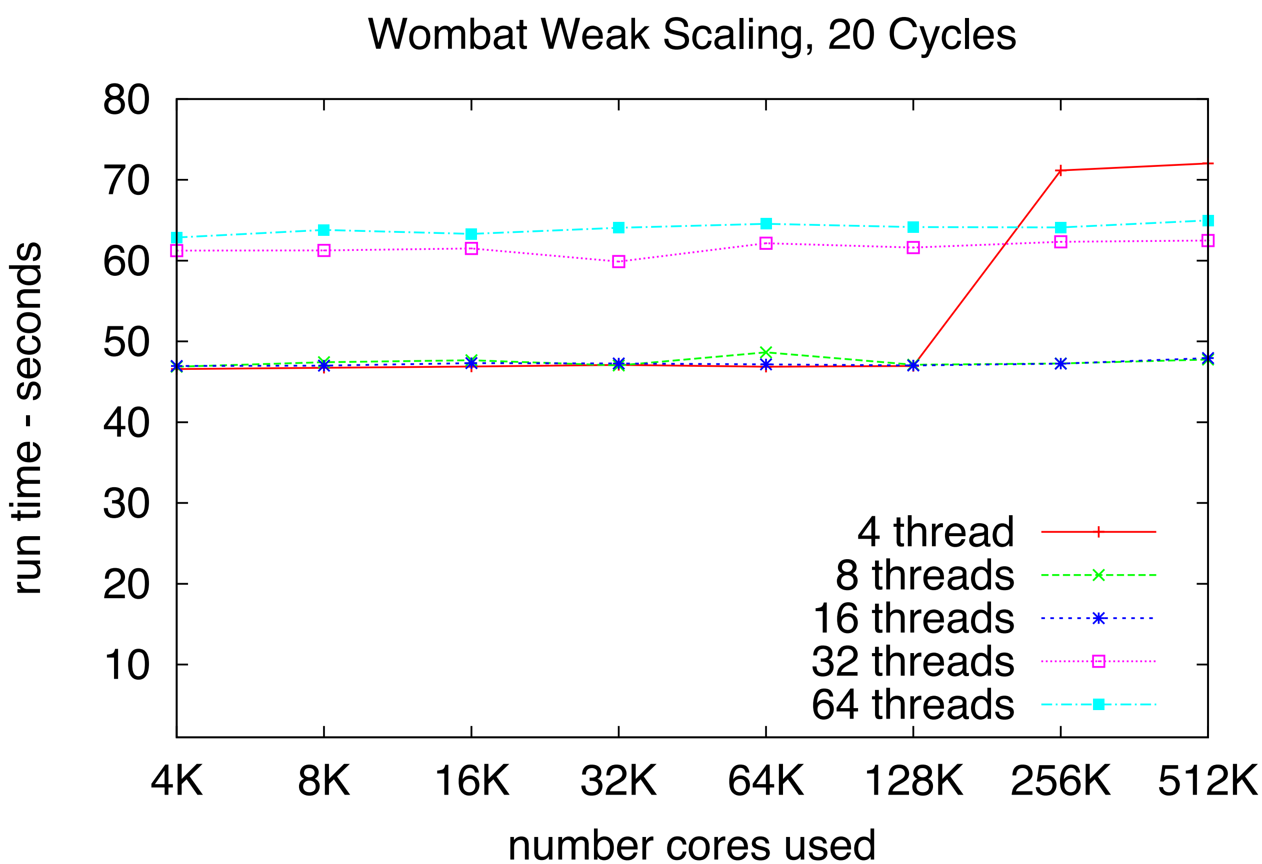


- Patches sizes to fit into L3 cache
- Threads within process work on patches concurrently

Wombat – patch exchange



Wombat weak scaling results



Cray XC KNL processors:

- 64 cores/node (no HT used)
- Quad/cache mode

Future Work

Future Work

- Validate that the OSC RDMA component can be used with GPU memory (target Coral Power9+Volta)
- Explore using MPI RMA from the GPU processors themselves (likely will involve OpenUCX)
- Explore using MPI RMA with NVM (e.g. NVMe-oF)
- MPI RMA and endpoints (?)

References

References

- M. G. F. Dosanjh, T. Groves, R. E. Grant, R. Brightwell and P. G. Bridges, "RMA-MT: A Benchmark Suite for Assessing MPI Multi-threaded RMA Performance," 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, 2016, pp. 550-559.
- P. J. Mendygral, N. Radcliffe, K. Kandalla, D. Porter, B. J. O’Neill, C. Nolting, P. Edmon, J M. F. Donnert, and T. W. Jones, “WOMBAT: A Scalable and High Performance Astrophysical MHD Code”, The Astrophysical Supplemental Journal Series, 2017, Vol 228(2).
- P. W. Howard and J. Walpole, “Relativistic Red-Black Trees”, Concurrency Computation: Pract. Exper., 2014, Vol 26, pages 2684–2712.
- R. Gestenberger, M. Besta, and T. Hoefler, “Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided”, 2013, IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC13).