Fast One-Sided Communication on Supercomputers and Application to Scientific Codes

Jeff R. Hammond[†] (`jhammond@alcf.anl.gov`), Sreeram Potluri[°], Cynthia Gu[♯], Alex Dickson[♭], Jim Dinan[†], Ivo Kabadshow[♮], Pavan Balaji[†], Vinod Tipparaju[*]

[†] Argonne National Laboratory [°] Ohio State University, [♯] Florida State University, [♭] The University of Chicago, [♮] Jülich Supercomputing Centre, [*] Oak Ridge National Laboratory

Motivation

Hardware architectural features

- One-sided RDMA support is ubiquitous.
- Noncontiguous operations enabled in hardware.
- Processor and network both possess significant concurrency (10- to 100-fold).
- Torus topology with adaptive routing (to avoid both hotspots and faults).

Communication middleware

- Message-passing ala MPI requires matching semantics.
- Portals 4 defaults to weak ordering.
- OpenSHMEM is unordered (fence required).
- GA/ARMCI stipulates strict p2p ordering (location consistency) of blocking operations.

In order to provide the best performance for the most common applications, vendors do not provide location consistency in hardware. On Blue Gene/P, Cray Gemini and Blue Gene/Q, enforcing ordering both limits per-message bandwidth *and* extra messages to flush the network.

Extensive analysis of three applications reveals that the one-sided applications built on top of Global Arrays (or ARMCI) do not require the level of memory consistency provided. By relaxing this consistency to the lowest level required for each application, we observe improved performance due to better alignment with hardware features.

Further performance is gained from a stripped down implementation of one-sided. Rather than provide all possible features, the implementation only delivers those that the application requires. In many cases, these can be selected dynamically at runtime.

Blue Gene/P Performance

The performance of OSPRI against ARMCI for noncontiguous (two-dimensional) Global Arrays communication is demonstrate below. Both local and remote communication is heavily optimized for message sizes relevant to applications.

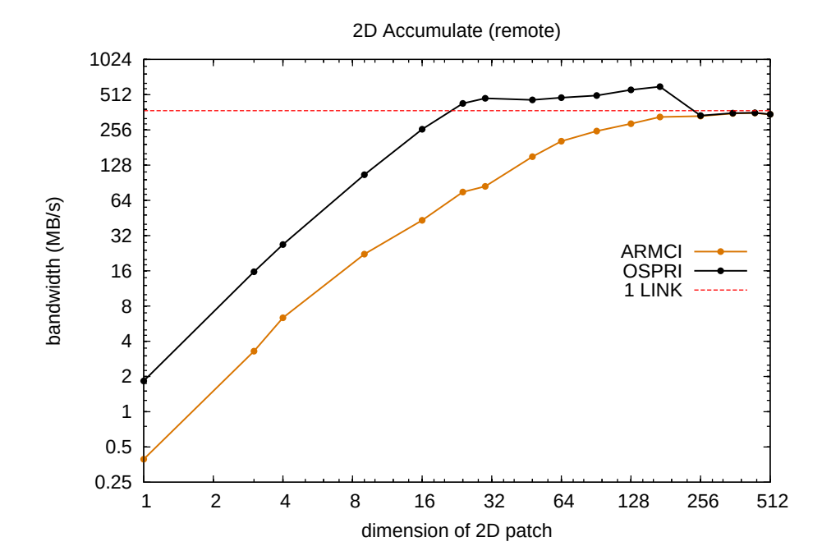


Figure: Remote GA accumulate.

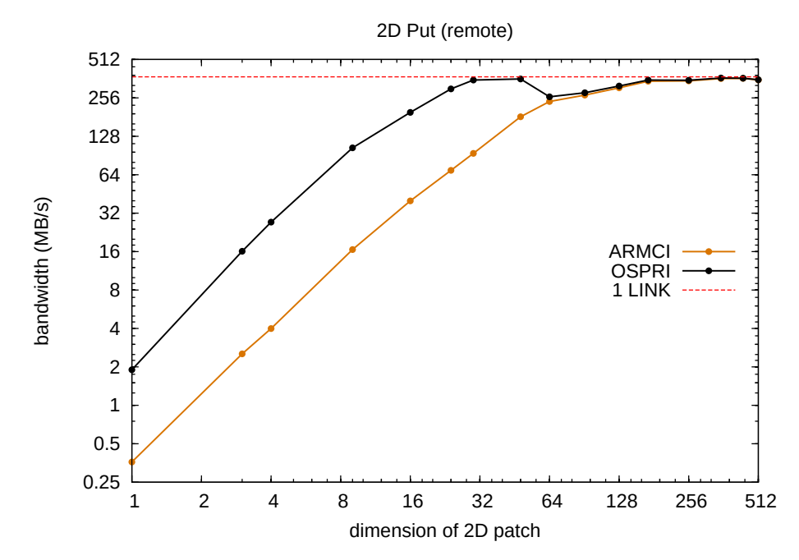


Figure: Remote GA put.

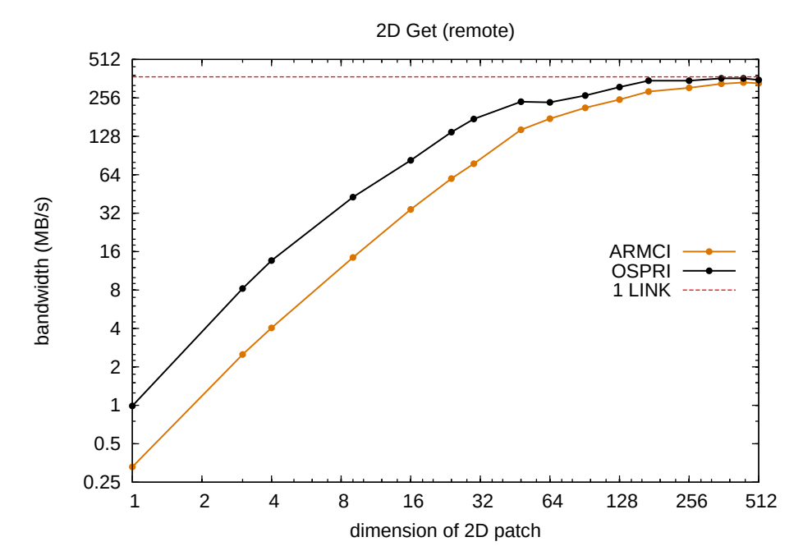


Figure: Remote GA get.

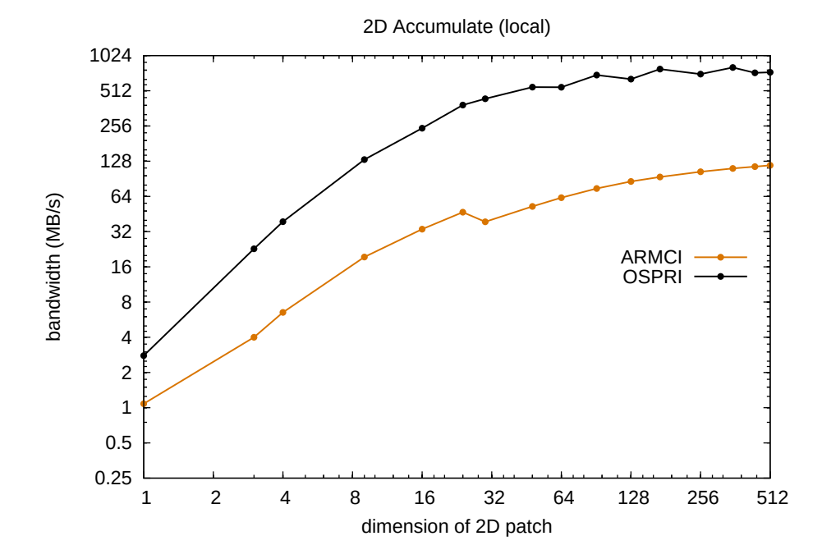


Figure: Local GA accumulate.

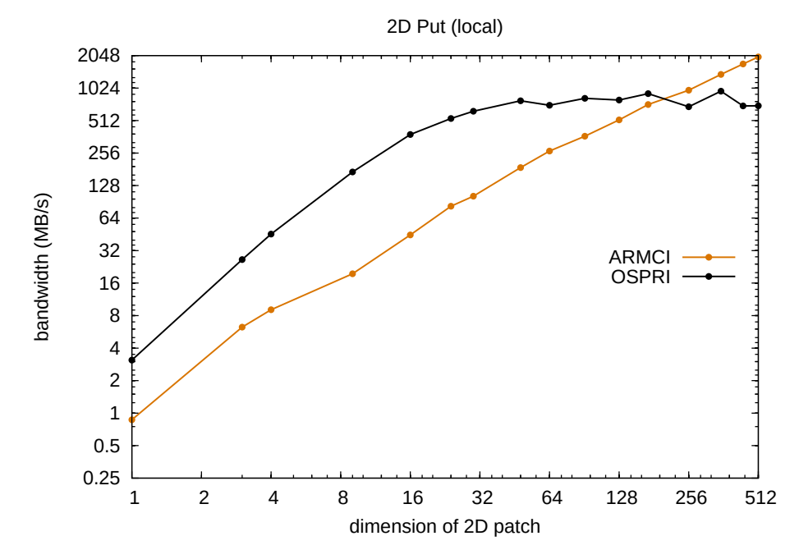


Figure: Local GA put.

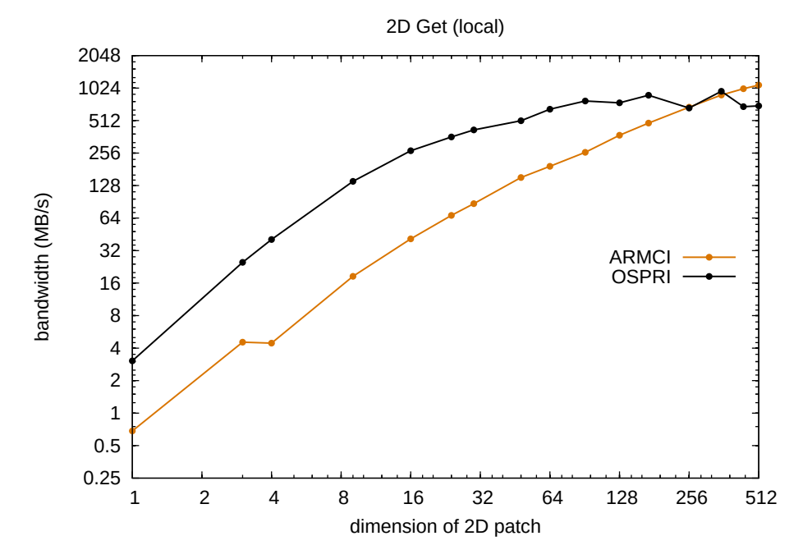


Figure: Local GA get.

The reason ARMCI is faster for large local put/get operations is that it offloads these to the DMA, but this consumes network resources that are better devoted to real (i.e. remote) communication.

OSPRI (One-Sided PRImitives)

OSPRI (One-Sided PRImitives) is a new one-sided communication runtime system targeting implementation of the PGAS programming (e.g. Global Arrays, UPC and CAF). It supports the key primitives of remote put, get and accumulate for contiguous (both blocking and non-blocking), strided and general noncontiguous input buffers, remote atomics (e.g. fetch-and-add and lock/unlock), fence and request-based synchronization and utility functions for managing memory. Unlike MPI-2 RMA, local and remote completion are treated seperately.

By design, this library is thread-safe and can utilize parallelism in the network controller. For example, the Blue Gene/Q PAMI API exposes parallelism in the MU via contexts, which allow multiple threads to initiate communication independently without a global mutex. Modern interconnects frequently do not provide ordering by default, so we support three different ordering modes so that application programmers can enforce ordering only when necessary in order to maximum performance.

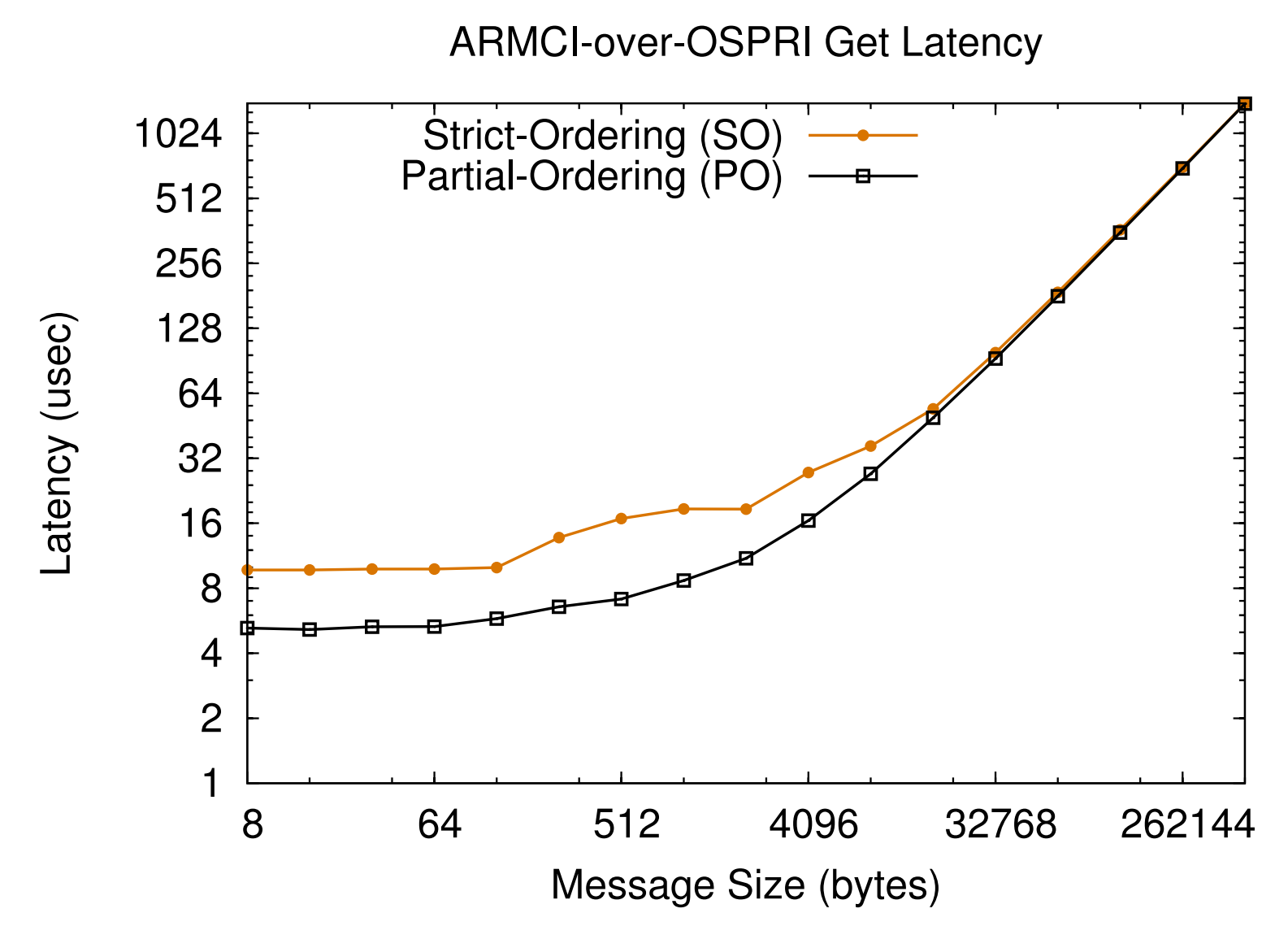


Figure: Effect of ordering semantics on latency (Blue Gene/P implementation).

As seen above, partial ordering improves latency significantly for messages up to 8 KB. Partial ordering means that each type of one-sided message is ordered with other messages of the same type, but not with other types of messages. For example, we order puts with other puts, but not with gets or accumulates.

See “OSPRI: An Optimized One-Sided Communication Runtime for Leadership-Class Machines” (ANL/MCS-P1880-0411[1] for a more complete description of OSPRI.)

[1] `http://www.mcs.anl.gov/publications/paper_detail.php?id=1545`

Cray Gemini

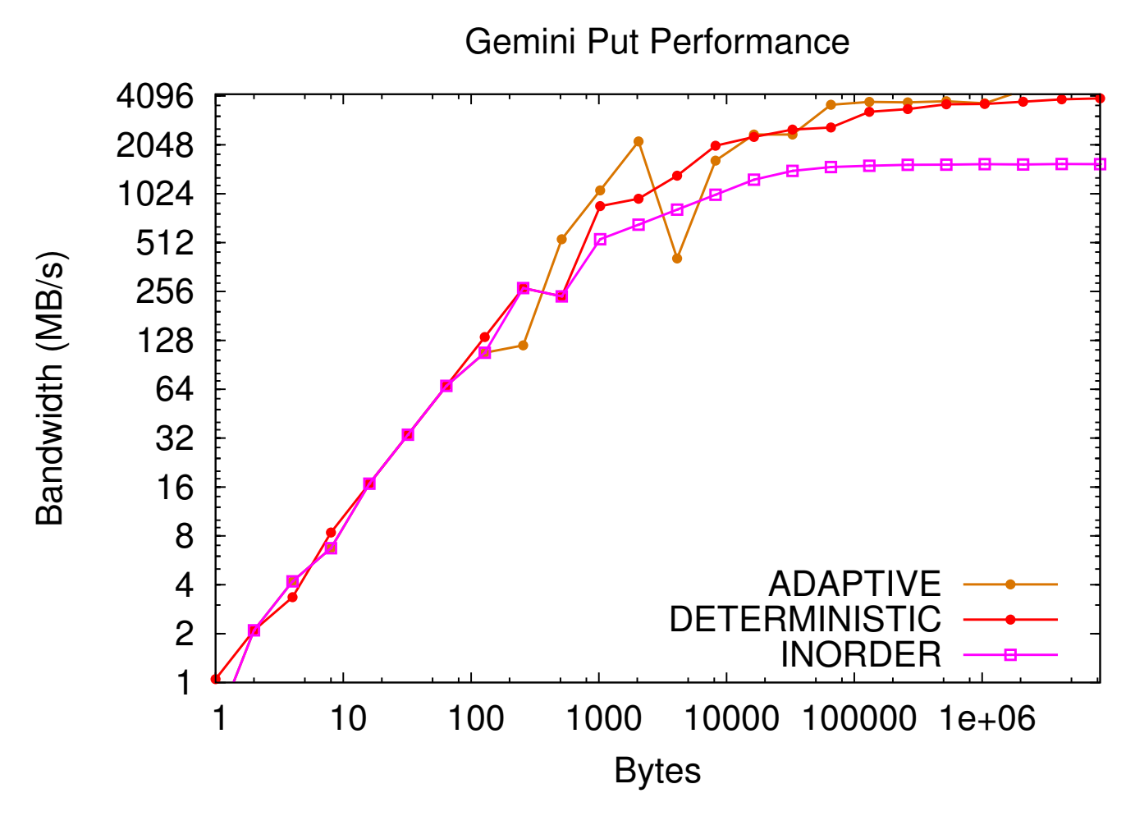


Figure: Effect of ordering semantics for the Gemini interconnect.

Application: ScaFaCoS

- The fast multipole method (FMM) is an exascale-oriented algorithm due to neighborhood communication, hierarchical structure and efficiency for very large problems.
- ScaFaCoS (developed at Jülich Supercomputing Centre) relies upon small to medium put operations and fast synchronization.
- The scaling of this implementation of FMM is limited if put latency is poor. The put latency of OSPRI on Blue Gene/P is an order-of-magnitude less than MPI-2 or ARMCI, enabling scaling to 300K cores.

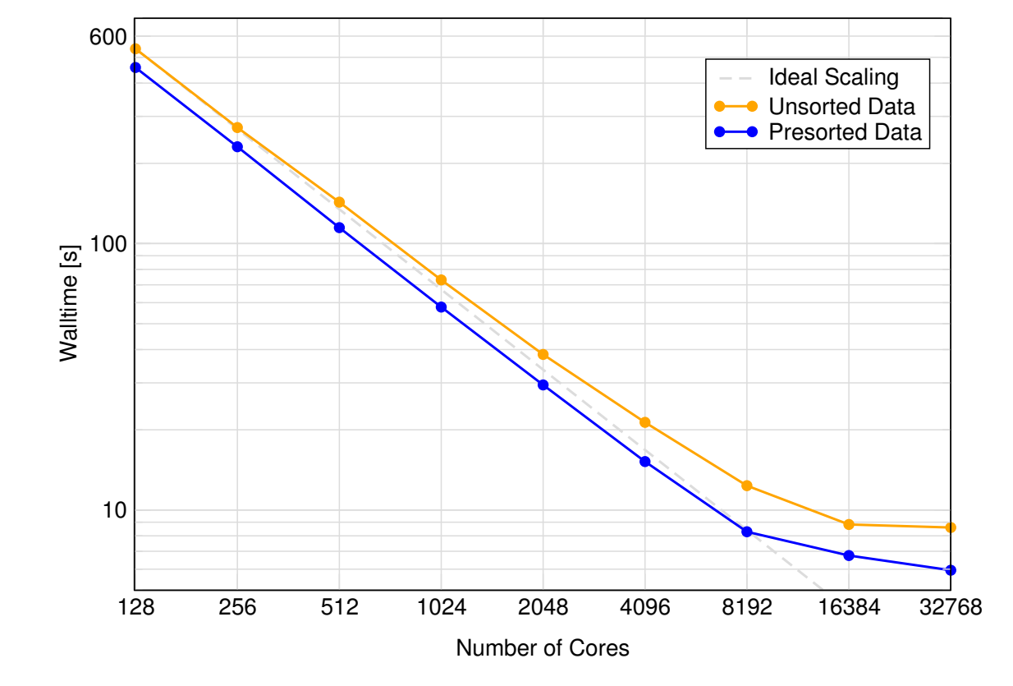


Figure: Strong-scaling FMM on Jugene.

		Time (s)	
Partition	Particles	Unsorted	Presorted
32768x1	1030607060301	3285	2203
73728x4	2010394559061	2288	530
73728x4	3011561968121	3812	715

Table: Trillion-particle FMM performance on Jugene with OSPRI.

		Time (s)	
Partition	Particles	ARMCI-MPI	OSPRI-DMAPP
168x24	1073741824	22.57	8.32

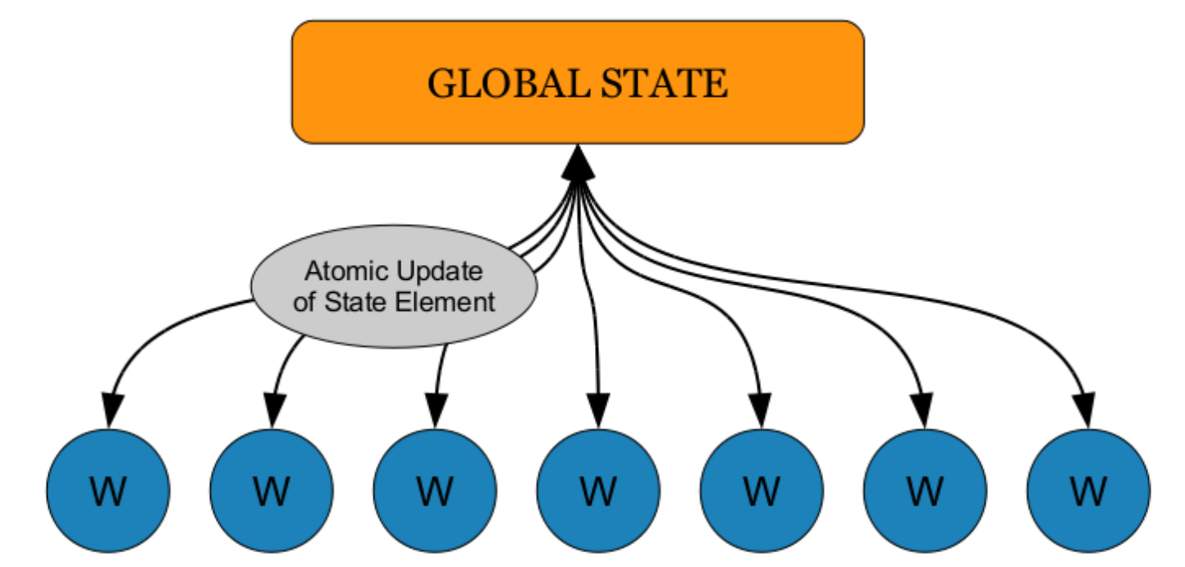
Table: Billion-particle FMM performance on Hopper with OSPRI.

Application: NEUS

Non-equilibrium umbrella sampling (NEUS) replaces a sequential single dynamics trajectory with many trajectories which can be computed in parallel and is thus a massively-parallel algorithm for the time dimension of biomolecular simulation.

NEUS is implemented using a global state updated by atomic (complex) operations asynchronously. The first implementation of NEUS used Global Arrays (GA) Put/Get operations in conjunction with Lock/Unlock provide a means to implement the necessary tools in a straightforward way.

One-sided communication allows a *distributed* global state wherein atomic updates can occur simultaneously except when locked. A master-worker approach would force unnecessary serialization of updates.



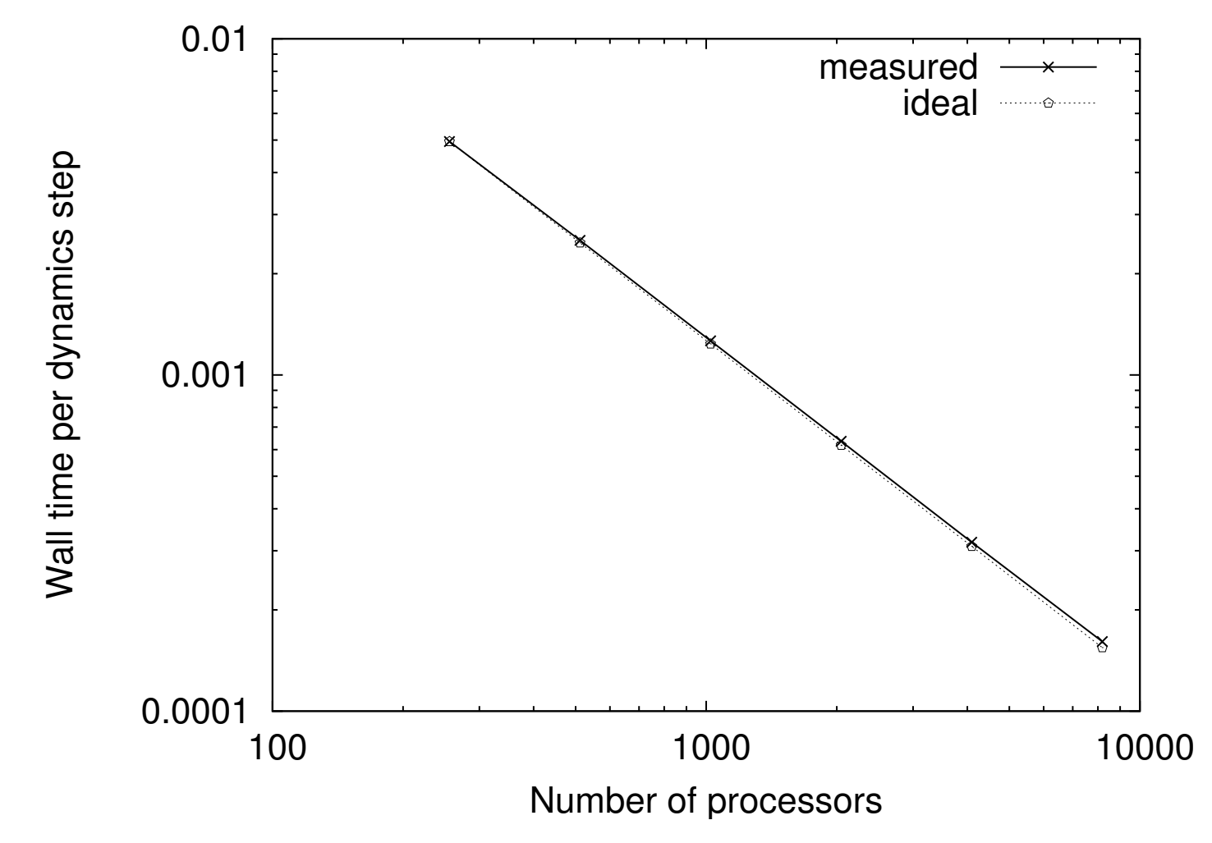


Figure: Strong scaling to 8192 cores of Blue Gene/P with 96% efficiency.

NEUS can be implemented with Global Arrays, MPI-2 RMA, PGAS languages and any other situation that provides asynchronous communication. Therefore, we can use this code to evaluate the implementation quality and semantic efficiency of such codes.