Implementation and Evaluation of OpenSHMEM Contexts Using OFI Libfabric

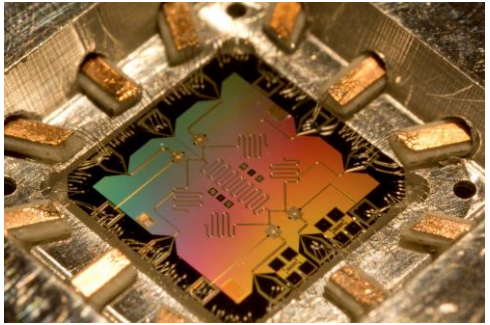
Max Grossman, Joseph Doyle, James Dinan, Howard Pritchard, Kayla Seager, Vivek Sarkar

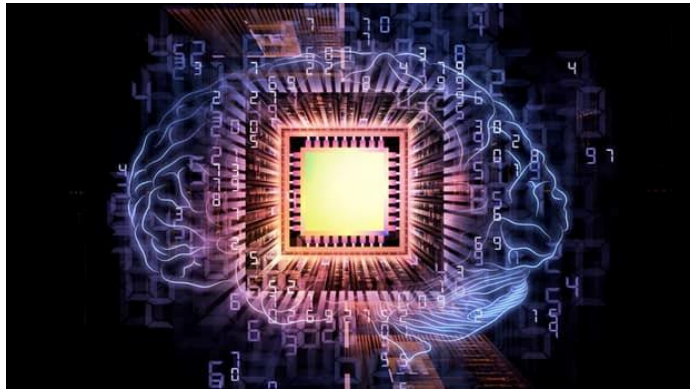
[1]Rice University [2]Carnegie Mellon University

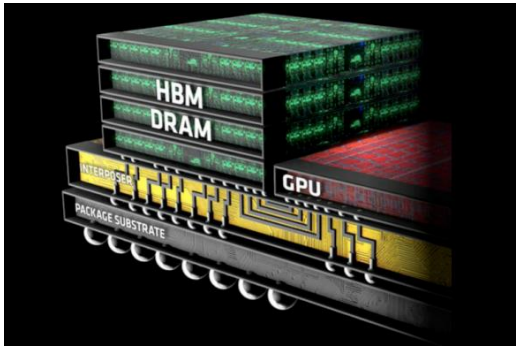
[3]Intel Corporation [4]Los Alamos National Laboratory

LA-UR-17-27150

Explosion in Processor Parallelism/Diversity







Renaissance in Parallel Programming Models

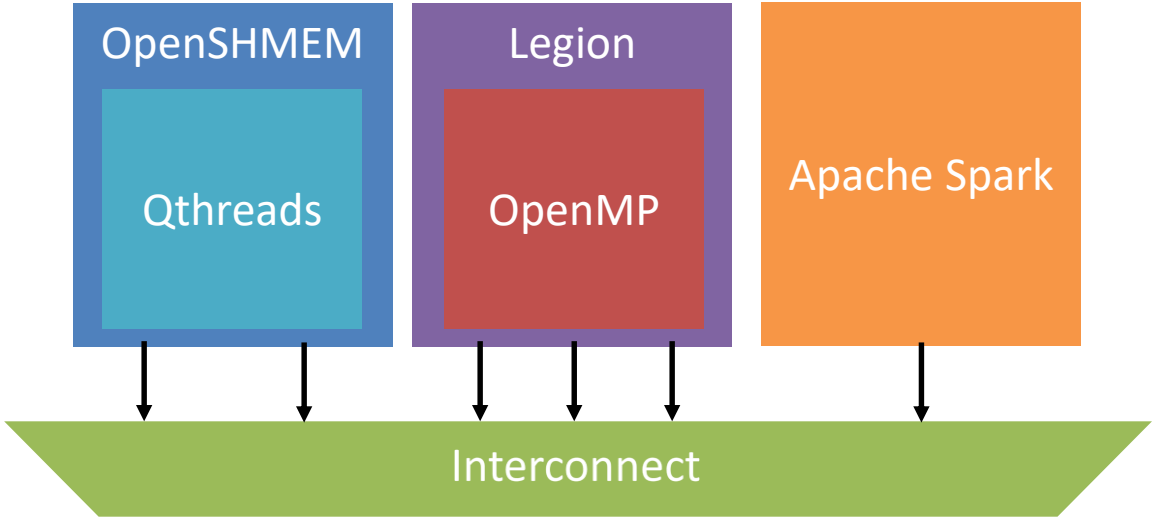
To match, also seen an explosion in programming models research/development for these new processor types:

- Kokkos
- Raja
- Qthreads
- Legion
- Parsec
- Habanero
- HiHAT
- hStreams
- CUDA
- OpenCL
- OCR
- UCX
- HPX
- Apache Spark
- ATMI
- ROCm
- HSA

Between a Rock and a Hard Place

Communication middleware then asked to flexibly support all of these new ways of programming HPC hardware.

Perform well for a variety of (often multi-tenant) workloads.



Communication Thread Safety to Date

Let the runtime do it (MPI-like thread safety).

Often the application/parallel runtime starts getting involved (e.g. AsyncSHMEM, Legion)

OpenSHMEM Contexts

Enables creation of separate logical endpoints into the network.

```
#pragma omp parallel
{
    shmem_ctx_t my_ctx;
    shmem_ctx_create(SHMEM_CTX_PRIVATE, &ctx);
    shmem_ctx_int_put(..., my_ctx);
}
```

Several benefits:

- Independent memory fences, quieting on different contexts
- Relaxation of thread safety on individual contexts reduces overheads
- Offers hints to runtime as to how to partition network resources
- Significant performance benefits at small to medium packet sizes

Contributions

Implementation of OpenSHMEM Contexts over libfabric.

Implementation of several benchmarks over this new contexts implementation (SoS micro-benchmarks, Graph500, GUPS, Mandelbrot, Pipelined Reduction).

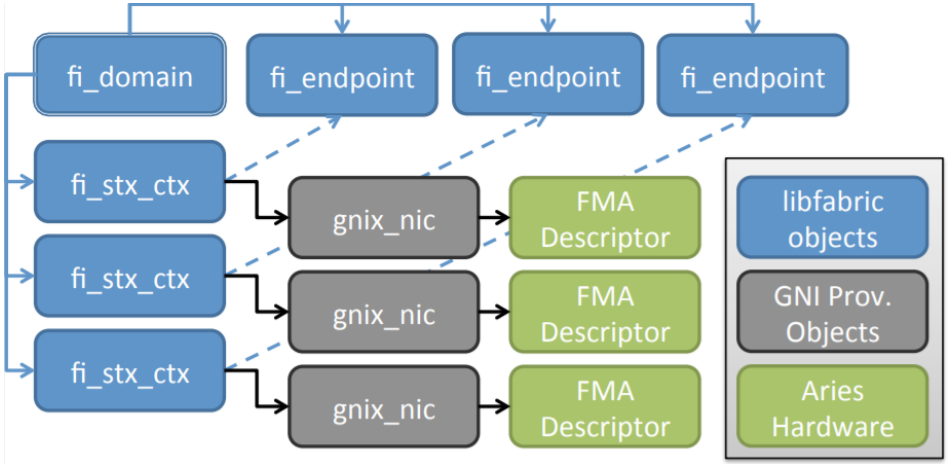
Evaluation of new benchmarks for performance and programmability using Edison and the Cray Aries interconnect.

libfabric

Platform independent, low-level library for high-performance interconnects.

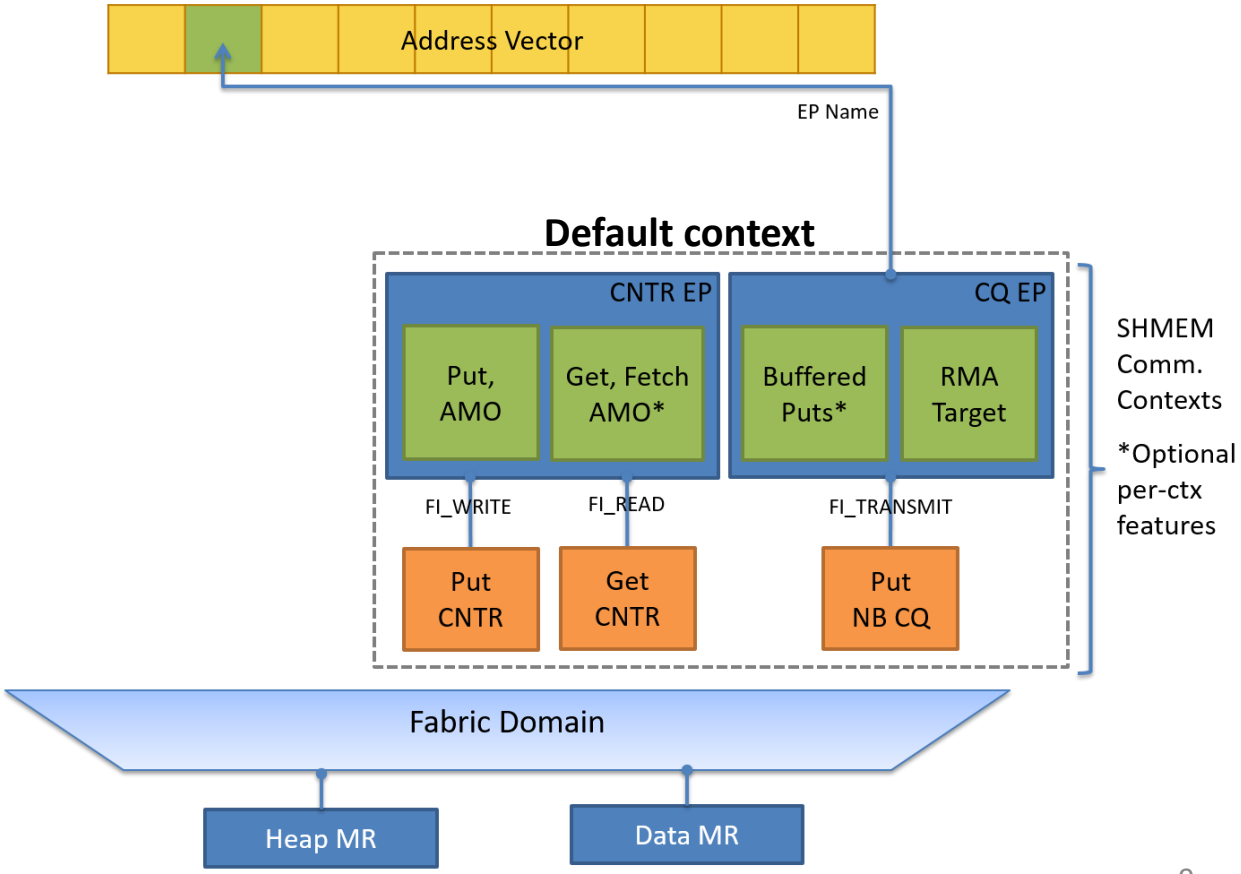
An `fi_domain` represents a hardware NIC.

- Encapsulates multiple `fi_endpoint` objects, represent hardware resources for sending/receiving messages
- `fi_stx_ctx` are bound to 1 or more `fi_endpoint` objects within an `fi_domain`, used to inject packets



Contexts on libfabric

For backwards compatibility, back the default SHMEM context with a single `fi_stx_ctx` bound to multiple `fi_endpoints`.

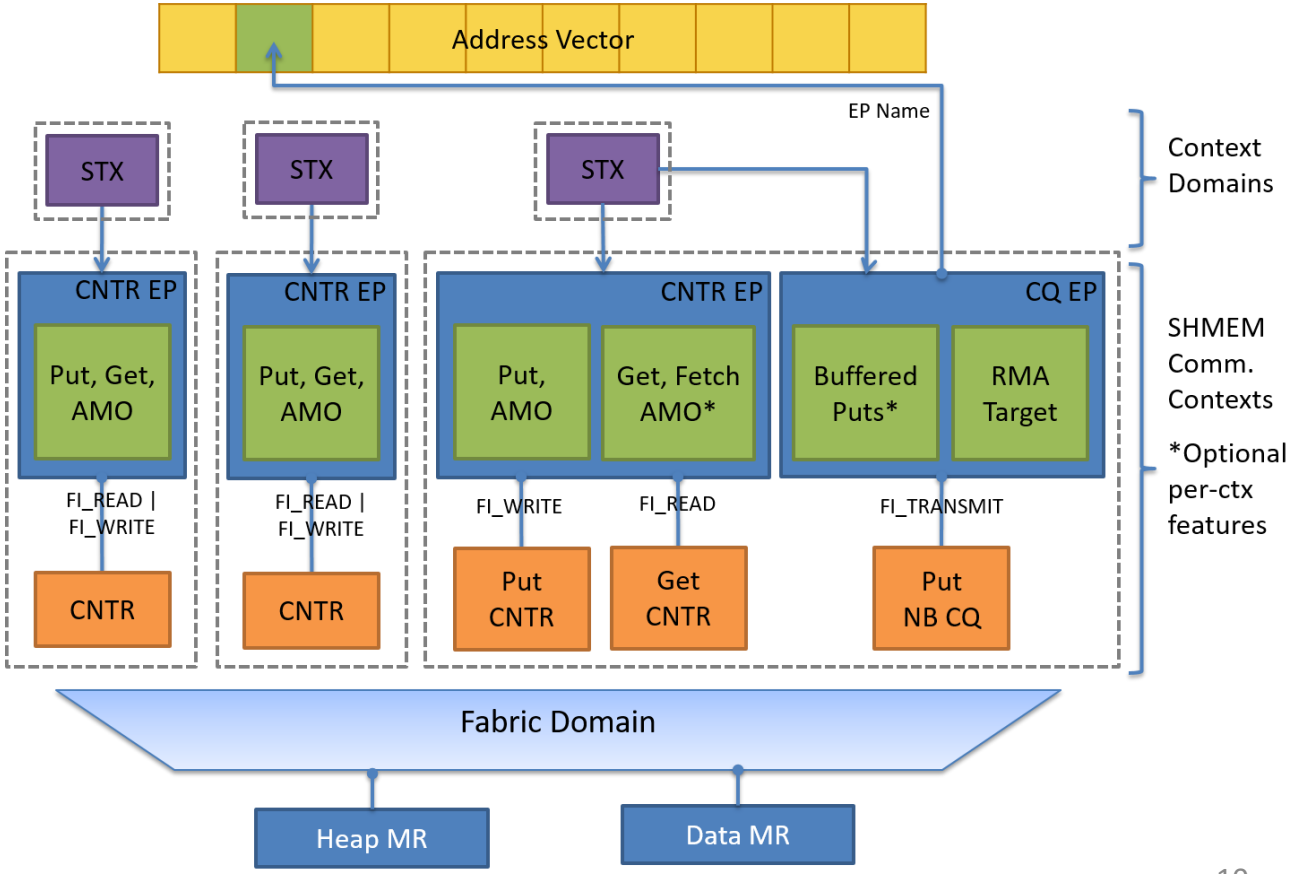


Contexts on libfabric

Associate explicitly allocated contexts with a single `fi_stx_ctx/fi_endpoint` pair.

Private resources reduce contention between contexts.

Single endpoint improves resource utilization.



Performance Evaluation

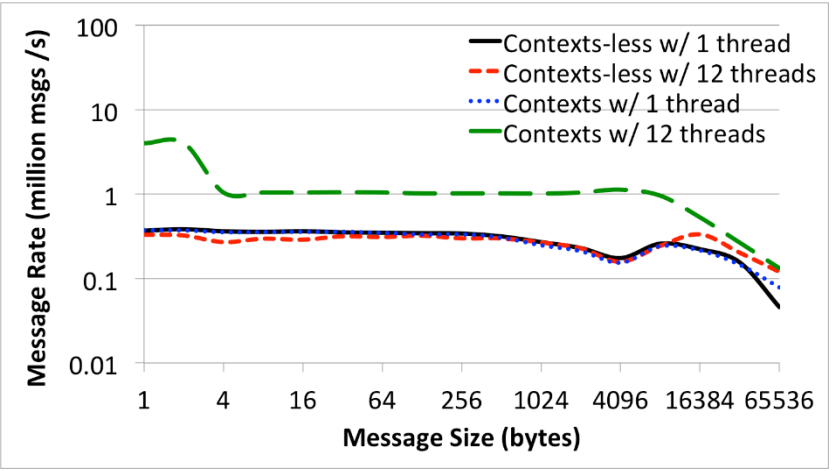
Implemented several benchmarks on top of libfabric-based contexts:

- Message rate, latency micro-benchmarks
- Graph500
- GUPS
- Mandelbrot
- Pipelined Reduction

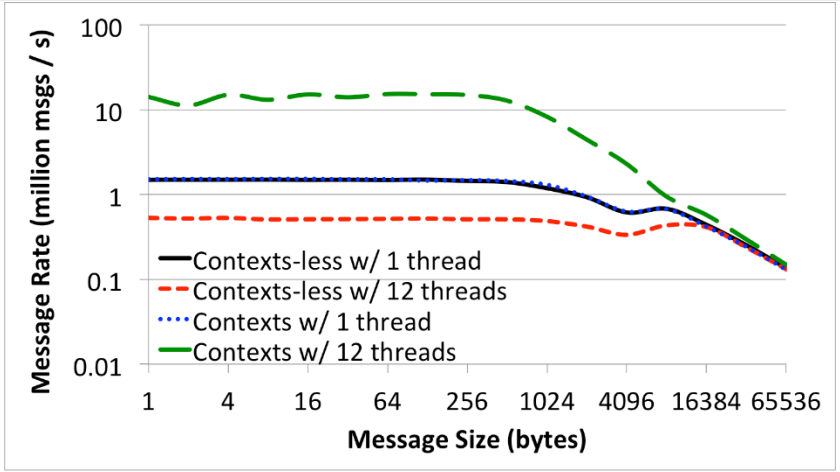
Experiments were run on Edison:

- 2 x 12-core Xeon + 64 GB DDR3 per node
- Cray Aries interconnect
- Flat tests run with 1 PE per core, hybrid tests run with 1 PE per socket and 12 threads per PE

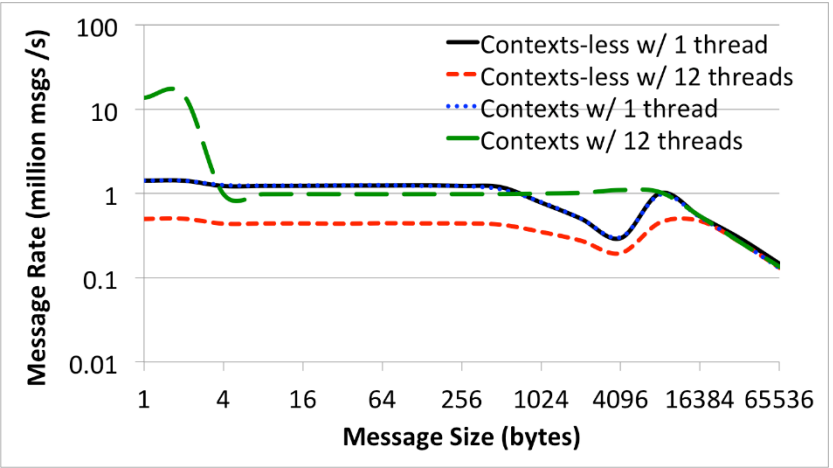
Contexts-Based Micro-Benchmarks



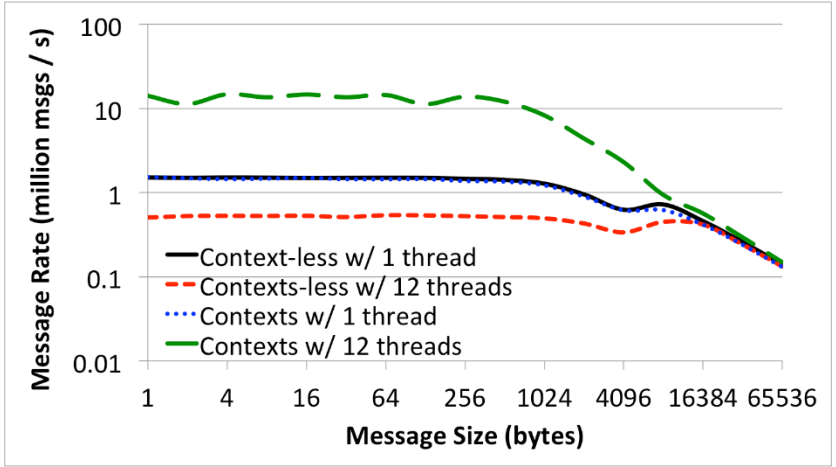
GET



PUT



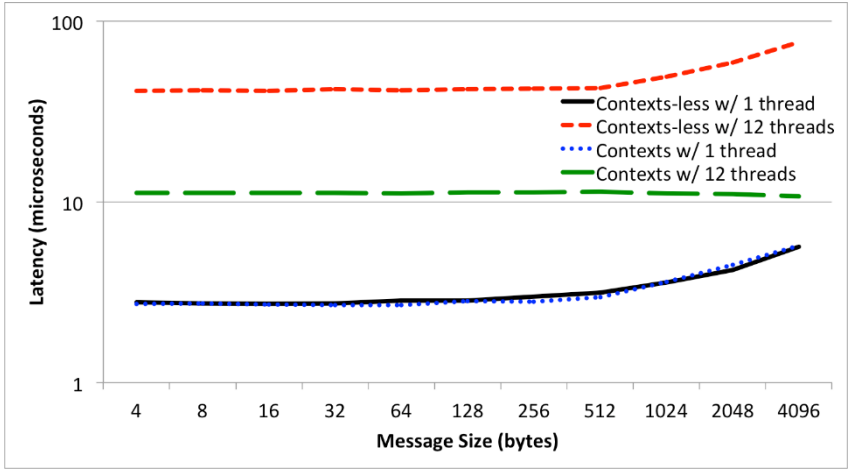
non-blocking GET



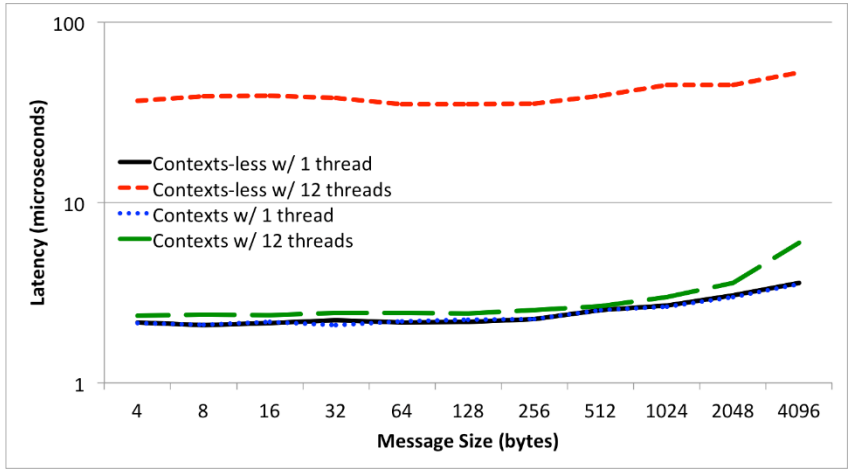
non-blocking PUT

Uni-directional message rates, not log-scale.

Contexts-Based Micro-Benchmarks



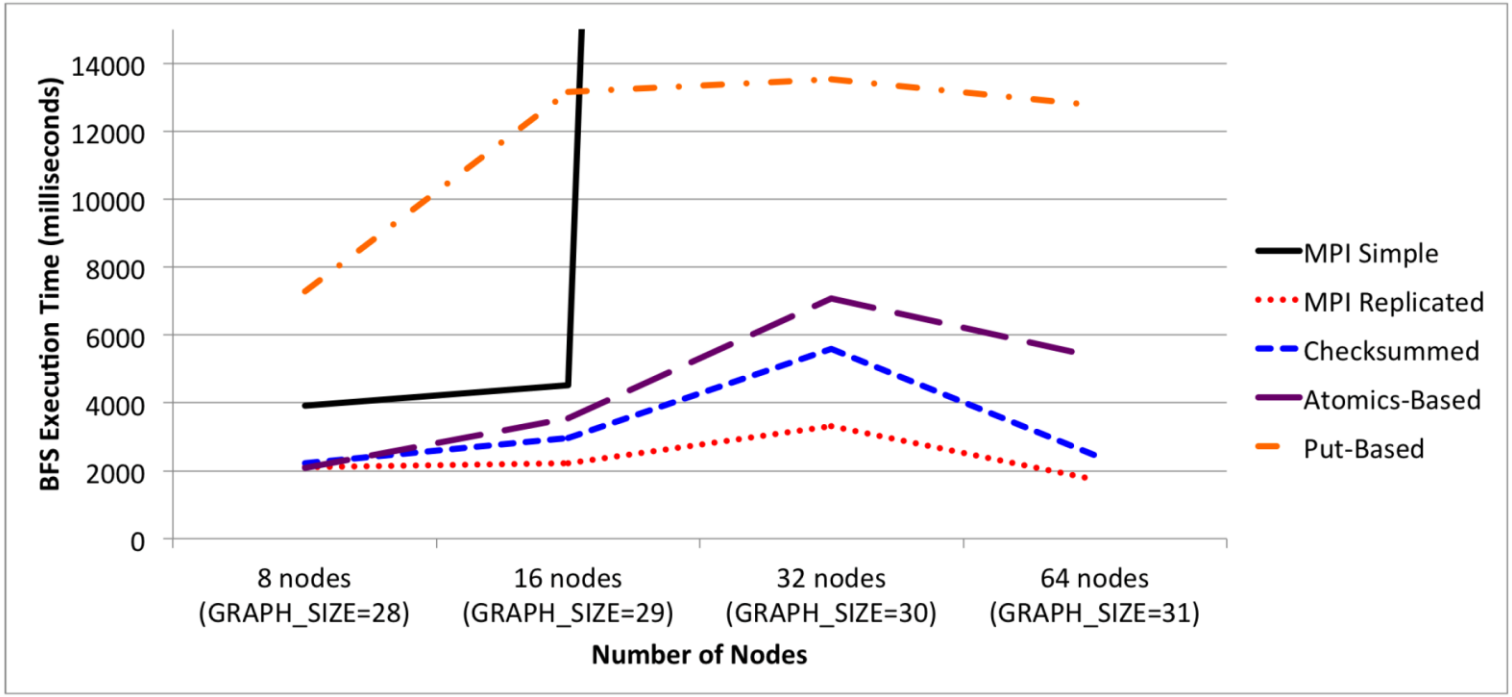
GET



PUT

Multi-threaded contention increases network latency (less so for contexts).

Contexts-Based G500



Contexts enabled profitable, programmable combination of multi-threading and OpenSHMEM (not achievable in earlier efforts).

Contexts-Based GUPS

API Used	Hybrid Parallelism?	Contexts Used?	Time (s)	Speedup
Gets/Puts	No	No	60.31	1.00x (Baseline)
Gets/Puts	Yes	No	1042.66	0.06x
Gets/Puts	Yes	Yes	330.08	0.18x
Bitwise Atomics	No	No	21.58	2.80x
Bitwise Atomics	Yes	No	306.23	0.20x
Bitwise Atomics	Yes	Yes	16.97	3.55x

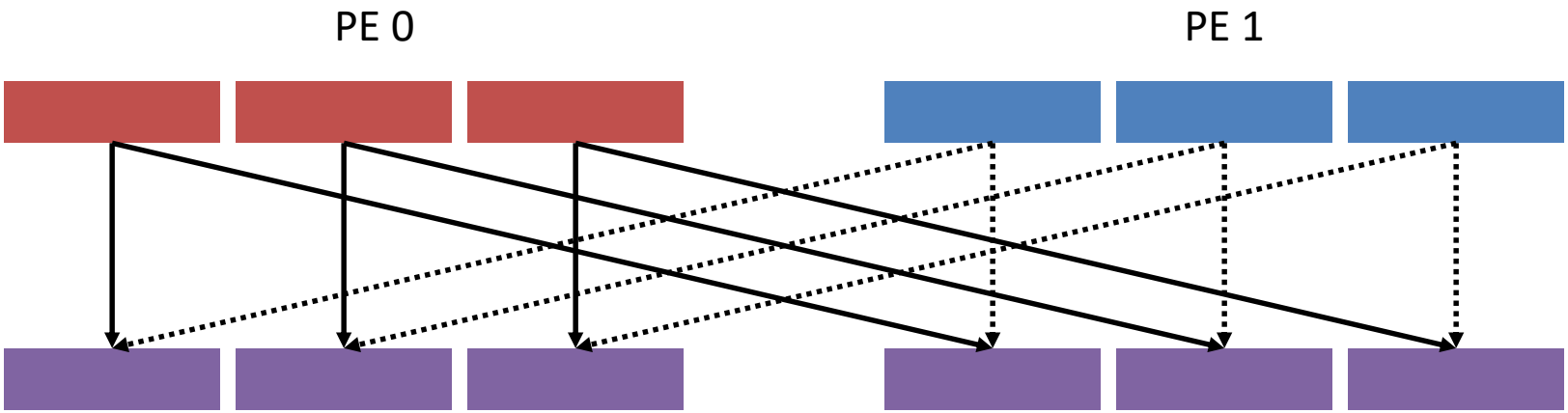
For small packet sizes, contexts continue to improve network utilization.
Big improvements from recent bitwise atomics extensions as well.

Contexts-Based Mandelbrot

Blocking APIs	Contexts?	Time (s)	Speedup
Yes	No	1,285.27	1.00x (Baseline)
Yes	For Multithreading	612.13	2.10x
Yes	Pipelined	592.56	2.17x
No	No	1,487.10	0.86x
No	For Multithreading	635.34	2.02x
No	Pipelined	608.20	2.11x

Use of one context per thread yields ~2x performance improvement, using contexts for pipelining improves performance by ~3%.

Contexts-Based Pipelined Reduction



Without Contexts	With Contexts
7.80 s	6.94 s

Single-threaded, performance improvement purely from communication pipelining

Programmability Evaluation

1. Improved network utilization at smaller packet sizes enables more algorithmically faithful implementations (less boiler plate, aggregating, etc).

2. Facilitation of hybrid programming has nice side effect of increasing state per PE, can help to reduce redundant/remote communication on many irregular applications (e.g. Graph500, UTS).

3. Did not find that creation/passing of contexts added significant development burden.

Contributions

Implementation of OpenSHMEM Contexts over libfabric.

Implementation of several benchmarks over this new contexts implementation (SoS micro-benchmarks, Graph500, GUPS, Mandelbrot, Pipelined Reduction).

Evaluation of new benchmarks for performance and programmability using Edison and the Cray Aries interconnect.

Acknowledgements

Los Alamos NATIONAL LABORATORY EST. 1943

DEPARTMENT OF DEFENSE UNITED STATES OF AMERICA

RICE

UNIVERSITY of HOUSTON

Stony Brook University