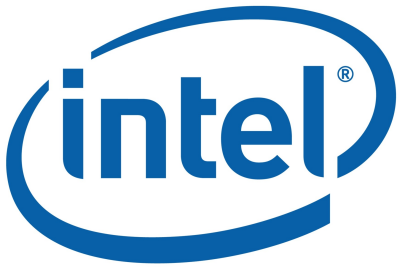
# Towards a matrix-oriented strided interface in OpenSHMEM

Jeff Hammond

Extreme Scalability Group & Parallel Computing Lab
Intel Corporation (Portland, OR)

7 October 2014



# Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.



**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

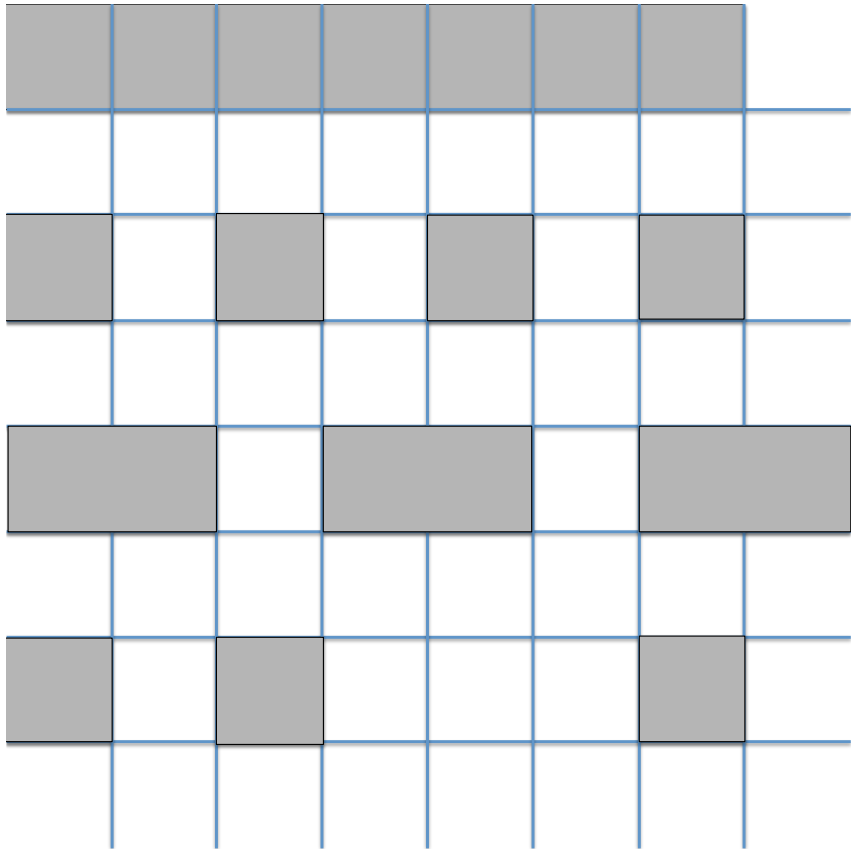
Notice revision #20110804

# Extreme Scalability Group Disclaimer

- I work in Intel Labs and therefore don't know anything about Intel products.
- I work for Intel, but I am not an official spokesman for Intel. Hence anything I say are my words, not Intel's. Furthermore, I do not speak for my collaborators, whether they be inside or outside Intel.
- You may or may not be able to reproduce any performance numbers I report.
- Performance numbers for non-Intel platforms were obtained by non-Intel people.
- Hanlon's Razor.

# Types of regular/direct data

- Contiguous
- Element Strided
- Block Strided
- Indexed



# Data types in APIs

- **Contiguous** - MPI, SHMEM, ARMCI, etc.
- **Element Strided** - MPI, SHMEM, ARMCI, etc.
- **Block Strided** - MPI, ~~SHMEM~~, ARMCI, etc.
- **Indexed** - MPI, ~~SHMEM~~, ARMCI, etc.

Where does it make sense to stop?

Why does SHMEM have element strided?

# Hardware considerations

- **Contiguous** - Obvious.
- **Element Strided** - Sub-packet elements bad.
- **Block Strided** - Amortize call overhead.
- **Indexed** - Where to stop with descriptors?

If SHMEM message-rate is high enough, all we need to do is exceed packet size.

**Except we have to wait on local completion for every call...**

# Datatypes vs. nonblocking

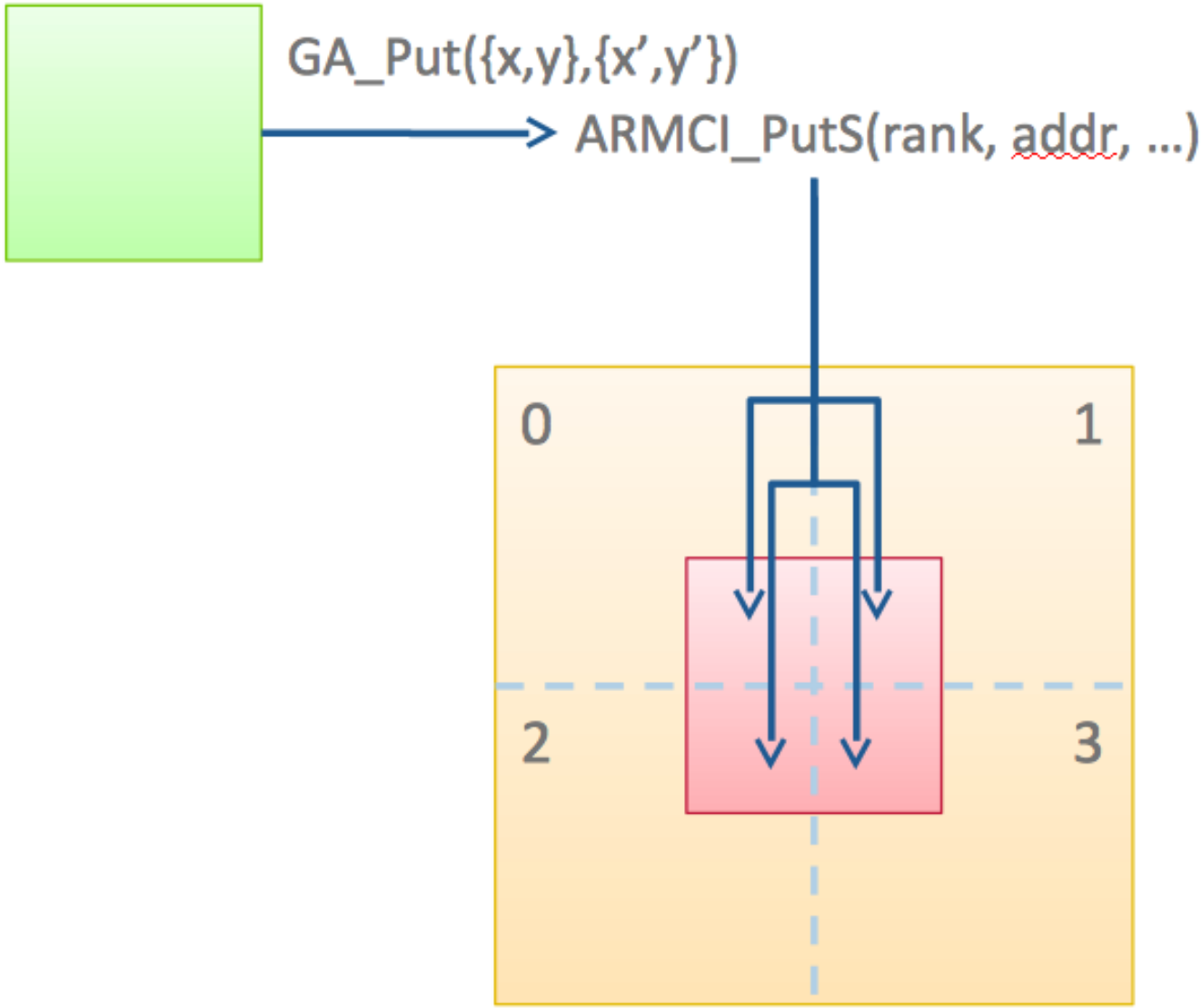
Nonblocking SHMEM Put and Get would seriously improve the situation, but one still has $O(N_{\mathrm{MIN}(rows,cols)})$ function calls for a submatrix.

Some networks (e.g. Cray Aries) don't let us inject an arbitrary number of nonblocking operations without a synchronization call.

Linear algebra entails a particular data semantic that deserves first-class treatment in the API.

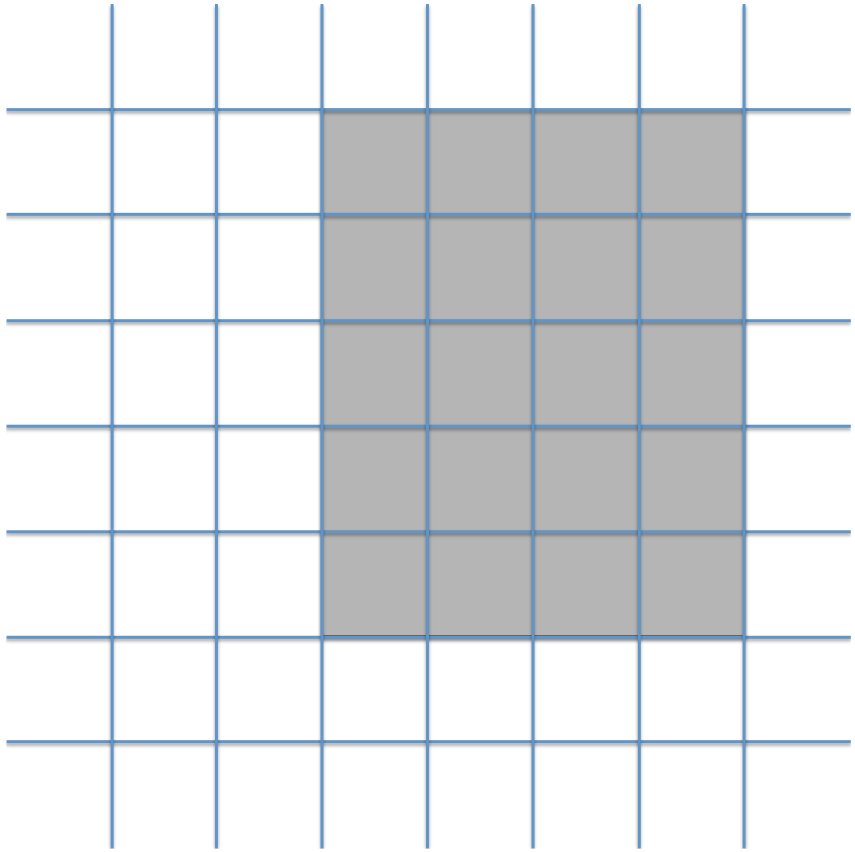
Runtime systems can do more with more...

# Global Arrays - (sub)matrices are first-class objects

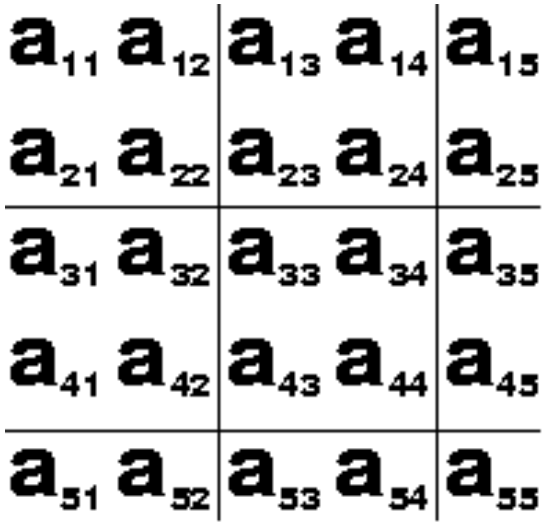


# Submatrix communication

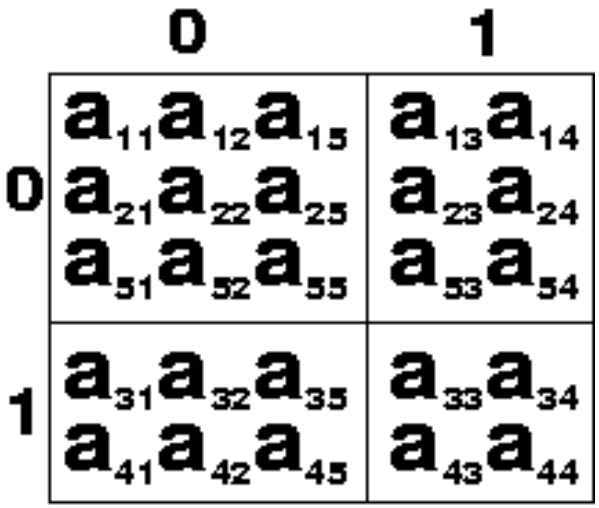
- Algorithmic block size may not match data block size.
- Data access pattern may change over the lifetime of object.
- If embedding complex data in a matrix, may want only one component.



# ScaLAPACK block cyclic



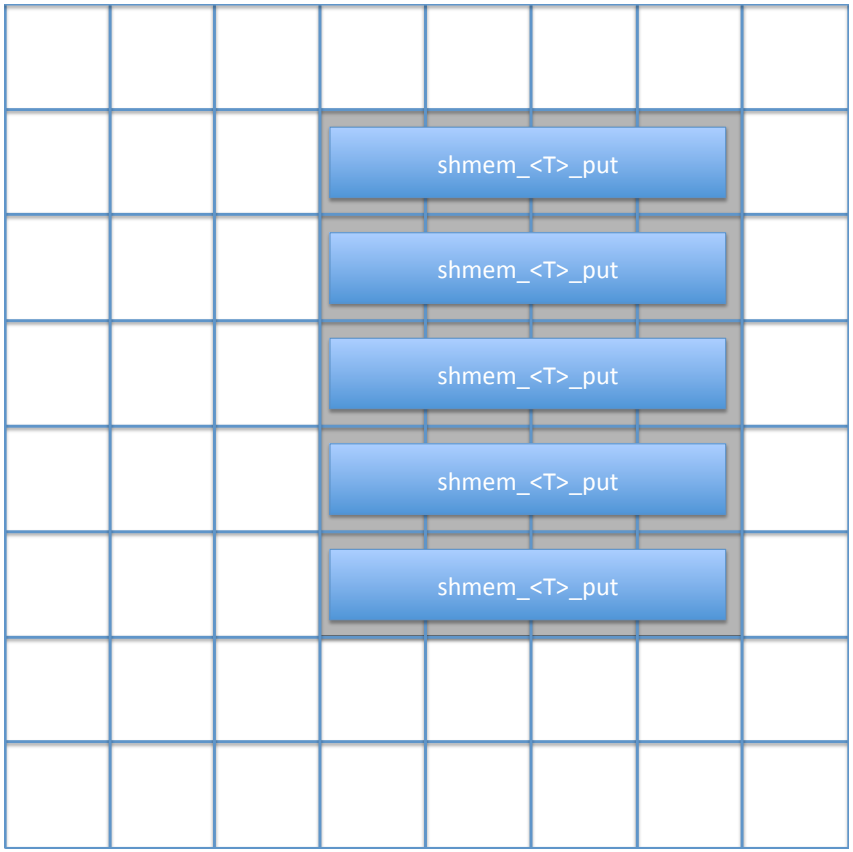

5 x 5 matrix partitioned in 2 x 2 blocks




2 x 2 process grid point of view

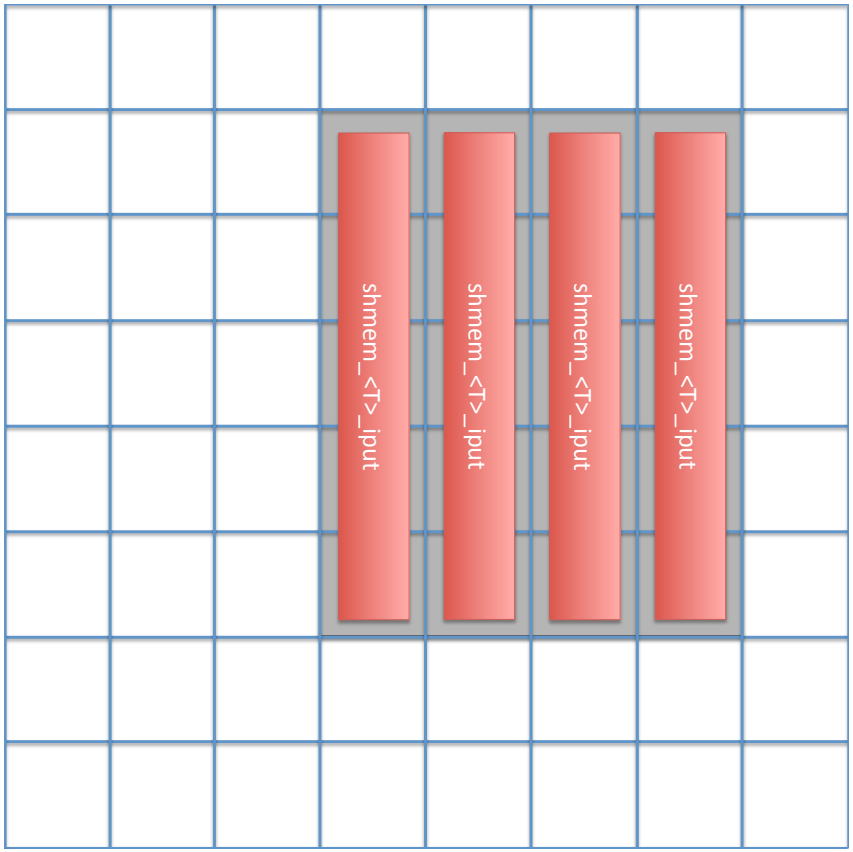
# Submatrix as many contiguous chunks

- Map subarray to vector of contiguous vectors.
- $O(N_{rows})$ function calls.
- Block until buffer accessible $\rightarrow$ bad for Get.
- This method far more reasonable with nonblocking.
- Efficient for $N_{cols} > N_{rows}$.



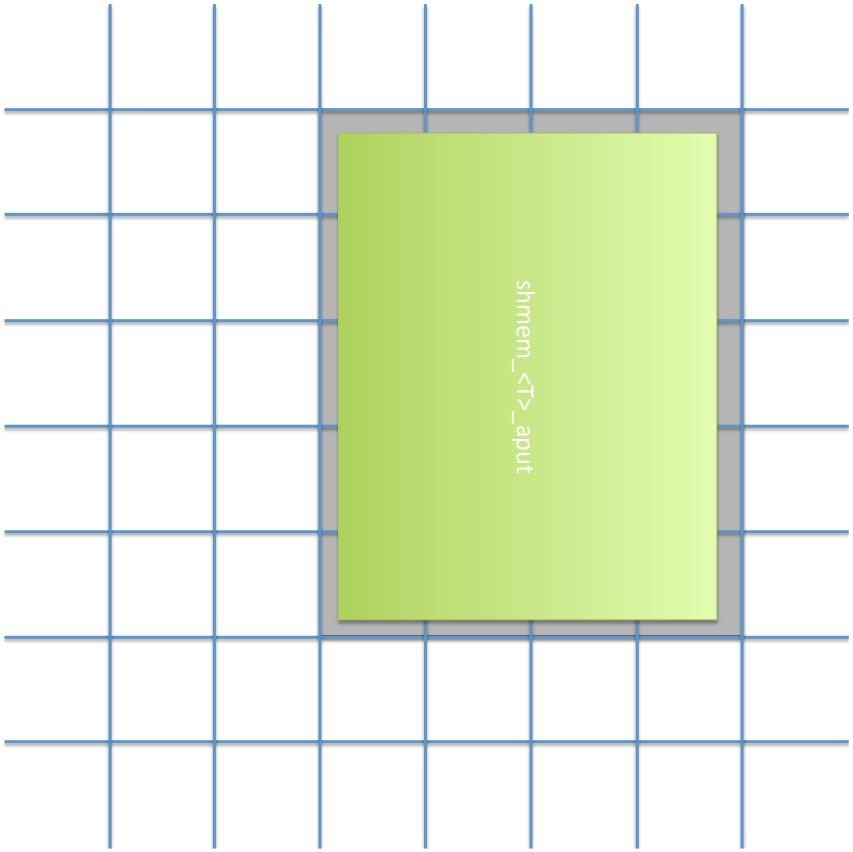
# Submatrix as many strided vectors

- Map subarray to vector of strided vectors.
- $O(N_{cols})$ function calls.
- Blocking bad here too.
- Strided touches same cache line repeatedly.
- Efficient for $N_{cols} < N_{rows}$?



# Submatrix communication

- One blocking function call.
- Runtime knows everything about data transfer.
- Maps directly to dense linear algebra semantics.



# Reference implementation

```
void shmemx_double_aput(double * dest, const double * src,
                        ptrdiff_t dstr, ptrdiff_t sstr,
                        size_t blksz, size_t blkct, int pe)
{
  double       *dtmp = dest;
  const double *stmp = src;
  if (blksz<blkct) /* may require tuning */ {
    for (size_t i=0; i<blksz; i++) {
      shmem_double_iput(dtmp, stmp, dstr, sstr, blkct, pe);
      dtmp++; stmp++;
    }
  } else {
    for (size_t i=0; i<blkct; i++) {
      shmem_double_put(dtmp, stmp, blksz, pe);
      dtmp += dstr; stmp += sstr;
    }
  }
}
```

# Optimized implementation

```
void shmemx_double_aput(double * dest, const double * src,
                        ptrdiff_t dstr, ptrdiff_t sstr,
                        size_t blksz, size_t blkct, int pe)
{
  int maxnbi = DMAPP_DEF_OUTSTANDING_NONBLOCKING/2;
  double       *dtmp = dest;
  const double *stmp = src;
  /* skipping iput implementation */
  for (size_t i=0; i<blkct; i++) {
      dmapp_put_nbi(dtmp, _sheap, pe, (double*)stmp,
                    blksz, DMAPP_QW);
      if (i && i%maxnbi==0) dmapp_gsync_wait();
      dtmp += dstr; stmp += sstr;
  }
  if (blkct%maxnbi!=0) dmapp_gsync_wait();
}
```

# Other optimized implementations

- Sreeram Potluri and I wrote a highly optimized implementation for IBM® Blue Gene/P* in DCMF* that packed up to packet granularity (active-message unpacking) and otherwise injected directly.
- PAMI* (Blue Gene/Q*, PERCS*) has a datatype engine that can in theory map to network DMA scatter-gather.
- Surely there is something for InfiniBand®...
- OSHMPI (MPI-3 RMA as conduit) maps directly to subarray type.
- Any one-sided API that has nonblocking should benefit.

*Other names and brands may be claimed as the property of others.

# Performance Results

- The performance improvement over a straightforward implementation on Blue Gene/P was huge. We never compared against SHMEM-like implementation and the hardware is all scrap now ☹
- Cray® XC30 tests at small scale show modest improvement. For small jobs (especially within an Aries quartet) without any contention, the benefit of nonblocking is not too large.

It will be a lot easier to gather performance data by mapping ARMCI to OpenSHMEM since we already have all the performance benchmarks oriented at Global Arrays usage.

# Acknowledgements

Jim Dinan (Intel) for an uncountable number of discussions of one-sided in SHMEM, ARMCI...

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.