Performance Advantages of Partitioned Global Address Space Languages

Katherine Yelick

U.C. Berkeley and Lawrence Berkeley National Laboratory

Joint work with


Motivation: Hardware Trends

• Communication latency has been relatively flat and will remain so
• Full bisection bandwidth will be increasingly difficult (meaning expensive) to obtain
• Number of cores per chip will roughly double every 2 years (possibly heterogeneous)

• As always, parallelism and locality will be key ... more than ever
Principles of Performance Software

To minimize the cost of communication

- **Use the best available communication mechanism** on a given machine
- **Hide communication** by overlapping
- **Avoid synchronization** using data-driven execution
- **Tune communication** using performance models when they work; search when they don’t
Best Available Communication Mechanism

- Data transfer (one-sided communication) is often faster than (two sided) message passing
- Semantics limit performance
  - In-order message delivery
  - Message and tag matching
  - Synchronization (message receipt) tied to data transfer
  - Need to acquire information from remote host processor

Part 1: Microbenchmarks
One-Sided vs Two-Sided: Theory

- A one-sided put/get message can be handled directly by a network interface with RDMA support
  - Avoid interrupting the CPU or storing data from CPU (preposts)
- A two-sided messages needs to be matched with a receive to identify memory address to put data
  - Offloaded to Network Interface in networks like Quadrics
  - Need to download match tables to interface (from host)
One-Sided vs. Two-Sided: Practice

- InfiniBand: GASNet vapi-conduit and OSU MVAPICH 0.9.5
- Half power point (N ½ ) differs by one order of magnitude
- This is not a criticism of the implementation!

Joint work with Paul Hargrove and Dan Bonachea
GASNet: Portability and High-Performance

GASNet better for latency across machines

Joint work with UPC Group; GASNet design by Dan Bonachea
GASNet: Portability and High-Performance

GASNet at least as high (comparable) for large messages

Joint work with UPC Group; GASNet design by Dan Bonachea
GASNet: Portability and High-Performance

GASNet excels at mid-range sizes: important for overlap

Joint work with UPC Group; GASNet design by Dan Bonachea
Hide communication by overlapping

- A programming model that decouples data transfer and synchronization
- Build remote address information into distributed data structures
- Application example

*Part 2: Partitioned Global Address Space Languages*
Partitioned Global Address Space

- **Global address space:** any thread/process may directly read/write data allocated by another
- **Partitioned:** data is designated as local (near) or global (possibly far); programmer controls layout

By default:
- Object heaps are shared
- Program stacks are private

- **3 Current languages:** UPC, CAF, and Titanium
- **3 New languages:** Chapel, Fortress, X10
**Bisection Bandwidth Limits**

- Full bisection bandwidth is (too) expensive
- During an all-to-all communication phase
  - Effective (per-thread) bandwidth is fractional share
  - Significantly lower than link bandwidth
  - Use smaller messages mixed with computation to avoid swamping the network
- Consider 3D FFT
  - 1D FFTs in each dimension, 3 phases
  - Transpose after first 2 for locality (1D partitioning)
Communication Strategies for 3D FFT

- **Three approaches:**
  - **Chunk:**
    - Wait for 2nd dim FFTs to finish
    - Minimize # messages
  - **Slab:**
    - Wait for chunk of rows destined for 1 proc to finish
    - Overlap with computation
  - **Pencil:**
    - Send each row as it completes
    - Maximize overlap and
    - Match natural layout

chunk = all rows with same destination

slab = all rows in a single plane with same destination

pencil = 1 row
NAS FT Variants Performance Summary

- Slab is always best for MPI; small message cost too high
- Pencil is always best for UPC; more overlap

<table>
<thead>
<tr>
<th>Component</th>
<th>#procs</th>
<th>MFlops per Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Myrinet</td>
<td>64</td>
<td>~360</td>
</tr>
<tr>
<td>Infiniband</td>
<td>256</td>
<td>~710</td>
</tr>
<tr>
<td>Elan3 256</td>
<td></td>
<td>~400</td>
</tr>
<tr>
<td>Elan3 512</td>
<td></td>
<td>~670</td>
</tr>
<tr>
<td>Elan4 256</td>
<td></td>
<td>~940</td>
</tr>
<tr>
<td>Elan4 512</td>
<td></td>
<td>~1170</td>
</tr>
</tbody>
</table>

- Best NAS Fortran/MPI
- Best MPI (always Slabs)
- Best UPC (always Pencils)

.5 Tflops
Avoid synchronization: Data-driven Execution

• Many algorithms require synchronization with remote processor
• Two possible mechanisms:
  • Signaling store: Raise a semaphore upon transfer
  • Remote enqueue: Put a task in a remote queue

Part 3: Event-Driven Execution Models
Avoid synchronization: Data-driven Execution

- Many algorithms require synchronization with remote processor
- What is the right mechanism in a PGAS model for doing this?
- Is it still one-sided?

*Part 3: Event-Driven Execution Models*
Mechanisms for Event-Driven Execution

- “Put” operation does a send side notification
  - Needed for memory consistency model ordering
- Need to signal remote side on completion
- Strategies:
  - Have remote side do a “get” (works in some algorithms)
  - Put + strict flag write: do a put, wait for completion, then do another (strict) put
  - Pipelined put + put-flag: works only on ordered networks
  - Signaling put: add new “store” operation that embeds signal (2<sup>nd</sup> remote address) into single message
Mechanisms for Event-Driven Execution

Signalling Store

- memput signal
- upc memput + strict flag
- pipelined memput (unsafe)
- MPI n-byte ping/pong
- MPI send/rec

Preliminary results on Opteron + Infiniband
Matrix Factorization

Completed part of U

A(i,j)  A(i,k)

Completed part of L

A(j,i)  A(j,k)

Trailing matrix to be updated

Panel factorizations involve communication for pivoting

Blocks 2D block-cyclic distributed

Matrix-matrix multiplication used here. Can be coalesced

Joint work with Parry Husbands
Communication Strategies for Factorization

• Three approaches:
  • Organize in bulk-synchronous phases (ScaLAPACK):
    • Factor a block column, then perform updates
    • Relatively easy to understand/debug, but extra synchronization
  • Overlapping phases (HPL):
    • Work associated with on block column factorization can be overlapped
    • Parameter to determine how many (need temp space accordingly)
  • Event-driven multithreaded (UPC Linpack):
    • Each thread runs an event handler loop
    • Tasks: factorization (w/ pivoting), update trailing, update upper
    • Tasks my suspend (voluntarily) to wait for data, synchronization, etc.
    • Data moved with remote gets (synchronization built-in)
    • Must “gang” together for factorizations
    • Scheduling priorities are key to performance and deadlock avoidance
**UPC HP Linpack Performance**

### X1 UPC vs. MPI/HPL

- **GFlop/s**
  - 60
  - X1/64
  - X1/128

### Opteron Cluster UPC vs. MPI/HPL

- **GFlop/s**
  - Opt/64

### Altix UPC. Vs. MPI/HPL

- **GFlop/s**
  - Alt/32

### UPC vs. ScaLAPACK

- **GFlops**
  - 2x4 proc grid
  - 4x4 proc grid

- Comparable to HPL (numbers from HPCC database)
- Faster than ScaLAPACK due to less synchronization
- Large scaling of UPC code on Itanium/Quadrics (Thunder)
  - 2.2 TFlops on 512p and 4.4 TFlops on 1024p

Joint work with Parry Husbands
Automatic Performance Tuning

- As with MPI, collective performance is key
- Developing collectives implementation using signaling store
- Automatic tuning (models and search)
- Compiler adds opportunity for global optimizations
What About MPI One-Sided?

• The performance advantages of one-sided communication should carry over to MPI
  • Missing from our study due to lack of time
  • Historically, not as well tuned as one would like

• MPI is not a suitable target for PGAS implementations
  • Language semantics do not restrict accesses to shared memory, i.e., no notion of epochs at the language level
  • Concurrency limits on access to window's memory would require sophisticated compiler analysis

example of prohibited behavior:

<table>
<thead>
<tr>
<th>window on P0</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
</table>

RMA_Put from P1  
local store from P0
Conclusions

• 1-sided communication can be faster than 2-sided
  • Decouples synchronization from data movement
  • Only pay for synchronization when you need it
  • Good match to RDMA support in networks
  • Very good for shared memory hardware

• PGAS languages
  • Source-to-source is the way to ubiquity
  • Complement highly tuned machine-specific compilers

• Common one-sided layer would be useful!
Communication in PGAS Languages

• Hiding communication costs will be key as machines scale
  • Especially if bisection bandwidth is limited

• Key features
  • Enable overlap with low overhead communication
  • Do not waste your network by doing maximum aggregation
  • Avoid packing overheads by communicating natural data structures

• Use multithreading for algorithms with complex dependencies
  • E.g., Matrix factorization in UPC

• Status:
  • Dense LU done: HPL-compliant
  • Sparse version (starting with Cholesky) underway

Joint work with Parry Husbands
Arrays in a Global Address Space

- **Key features of Titanium arrays**
  - Generality: indices may start/end and any point
  - Domain calculus allow for slicing, subarray, transpose and other operations without data copies

- **Use domain calculus to identify ghosts and iterate:**
  ```c
  foreach (p in gridA.shrink(1).domain()) ...
  ```

- **Array copies automatically work on intersection**
  ```c
  gridB.copy(gridA.shrink(1));
  ```

Useful in grid computations including AMR

Joint work with Titanium group
AMR in Titanium

C++/Fortran/MPI AMR
- Chombo package from LBNL
- Bulk-synchronous comm:
  - Pack boundary data between procs

Titanium AMR
- Entirely in Titanium
- Finer-grained communication
  - No explicit pack/unpack code
  - Automated in runtime system

<table>
<thead>
<tr>
<th>Code Size in Lines</th>
<th>C++/F/MPI</th>
<th>Titanium</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMR data Structures</td>
<td>35000</td>
<td>2000</td>
</tr>
<tr>
<td>AMR operations</td>
<td>6500</td>
<td>1200</td>
</tr>
<tr>
<td>Elliptic PDE solver</td>
<td>4200*</td>
<td>1500</td>
</tr>
</tbody>
</table>

* Somewhat more functionality in PDE part of Chombo code

10X reduction in lines of code!

AMR Work by Tong Wen and Philip Colella
**Performance of Titanium AMR**

- **Serial:** Titanium is within a few % of C++/F; sometimes faster!
- **Parallel:** Titanium scaling is comparable with generic optimizations
  - optimizations (SMP-aware) that are not in MPI code
  - additional optimizations (namely overlap) not yet implemented

Joint work with Tong Wen, Jimmy Su, Phil Colella
Performance and Scalability

• Codes written in high level object-oriented PGAS language (Titanium)
  • Performing well on O(100) processors
  • Languages allow for hand-tuning
  • Automatic optimizations eliminate need for some hand-tuning and permits machine-specific optimizations

• Why you should care about PGAS languages on Petascale systems
  • Low overhead, one-sided communication
  • Expose the best performance of the hardware
  • Potentially faster than two-sided MPI