Evaluating the Impact of Programming Language Features on the Performance of Parallel Applications on Cluster Architectures

Konstantin Berlin\textsuperscript{1}, Jun Huan\textsuperscript{2}, Mary Jacob\textsuperscript{3}, Garima Kochhar\textsuperscript{3}, Jan Prins\textsuperscript{2}, Bill Pugh\textsuperscript{1}, P. Sadayappan\textsuperscript{3}, Jaime Spacco\textsuperscript{1}, Chau-Wen Tseng\textsuperscript{1}

\textsuperscript{1} University of Maryland, College Park
\textsuperscript{2} University of North Carolina, Chapel Hill
\textsuperscript{3} Ohio State University
Motivation

- Irregular, fine-grain remote accesses
  - Several important applications
  - Message passing (MPI) is inefficient

- Language support for fine-grain remote accesses?
  - Less programmer effort than MPI
  - How efficient is it on clusters?
Contributions

• Experimental evaluation of language features
• Observations on programmability & performance
• Suggestions for efficient programming style
• Predictions on impact of architectural trends

Findings not a surprise, but we quantify penalties for language features for challenging applications
Outline

• Introduction
• Evaluation
  – Parallel paradigms
  – Fine-grain applications
  – Performance
• Observations & recommendations
• Impact of architecture trends
• Related work
Parallel Paradigms

• **Shared-memory**
  – Pthreads, Java threads, OpenMP, HPF
  – Remote accesses same as normal accesses

• **Distributed-memory**
  – MPI, SHMEM
  – Remote accesses through explicit (aggregated) messages
  – User distributes data, translates addresses

• **Distributed-memory with special remote accesses**
  – Library to copy remote array sections (Global Arrays)
  – Extra processor dimension for arrays (Co-Array Fortran)
  – Global pointers (UPC)
  – Compiler / run-time system converts accesses to messages
Global Arrays

• Characteristics
  – Provides illusion of shared multidimensional arrays
  – Library routines
    • Copy rectangular shaped data in & out of global arrays
    • Scatter / gather / accumulate operations on global array
  – Designed to be more restrictive, easier to use than MPI

• Example

NGA_Access(g_a, lo, hi, &table, &ld);
for (j = 0; j < PROCS; j++) {
  for (i = 0; i < counts[j]; i++) {
    table[index-lo[0]] ^= stable[copy[i] >> (64-LSTSIZE)];
  }
}

NGA_Release_update(g_a, lo, hi);
UPC

• Characteristics
  – Provides illusion of shared one-dimensional arrays
  – Language features
    • Global pointers to cyclically distributed arrays
    • Explicit one-sided msgs (upc_memput(), upc_memget())
  – Compilers translate global pointers, generate communication

• Example

```c
shared unsigned int table[TABSIZE];
for (i=0; i<NUM_UPDATES/THREADS; i++) {
    int ran = random();
    table[ (ran & (TABSIZE-1)) ] ^= stable[ (ran >> (64-LSTSIZE)) ];
}
barrier();
```
UPC

- Most flexible method for arbitrary remote references
- Supported by many vendors
- Can cast global pointers to local pointers
  - Efficiently access local portions of global array
- Can program using hybrid paradigm
  - Global pointers for fine-grain accesses
  - Use upc_memput(), upc_mempget() for coarse-grain accesses
Target Applications

• **Parallel applications**
  – Most standard benchmarks are easy
    • Coarse-grain parallelism
    • Regular memory access patterns

• **Applications with irregular, fine-grain parallelism**
  – Irregular table access
  – Irregular dynamic access
  – Integer sort
Options for Fine-grain Parallelism

• Implement fine-grain algorithm
  – Low user effort, inefficient

• Implement coarse-grain algorithm
  – High user effort, efficient

• Implement hybrid algorithm
  – Most code uses fine-grain remote accesses
  – Performance critical sections use coarse-grain algorithm
  – Reduce user effort at the cost of performance
    • How much performance is lost on clusters?
Experimental Evaluation

- **Cluster : Compaq Alphaserver SC (ORNL)**
  - 64 nodes, 4-way Alpha EV67 SMP, 2 GB memory each
  - Single Quadrics adapter per node

- **SMP : SunFire 6800 (UMD)**
  - 24 processors, UltraSparc III, 24 GB memory total
  - Crossbar interconnect
Irregular Table Update

• Applications
  – Parallel databases, giant histogram / hash table

• Characteristics
  – Irregular parallel accesses to large distributed table
  – Bucket version (aggregated non-local accesses) possible

• Example

```cpp
for ( i=0; i<NUM_UPDATES; i++ ) {
    ran = random();
    table[ran & (TABSIZE-1)] ^= stable[ran >> (64-LSTSIZE)];
}
```
**Table Update (AlphaServer, $2^{22}$ table)**

- UPC / Global Array fine-grain accesses inefficient (100x)
- Hybrid coarse-grain (bucket) version closer to MPI
• UPC fine-grain accesses inefficient even on SMP
Irregular Dynamic Accesses

• Applications
  – NAS CG (sparse conjugate gradient)

• Characteristics
  – Irregular parallel accesses to sparse data structures
  – Limited aggregation of non-local accesses

• Example (NAS CG)
  
  for (j = 0; j < n; j++) {
    sum = 0.0;
    for (k = rowstr[j]; k < rowstr[j+1]; k++)
      sum = sum + a[k] * v[colidx[k]];
    w[j] = sum;
  }
• UPC fine-grain accesses inefficient (4x)
• Hybrid coarse-grain version slightly closer to MPI
Integer Sort

- **Applications**
  - NAS IS (integer sort)

- **Characteristics**
  - Parallel sort of large list of integers
  - Non-local accesses can be aggregated

- **Example** (NAS IS)

```c
for ( i=0; i<NUM_KEYS; i++ ) {    /* sort local keys into buckets */
    key = key_array[i];
    key_buff1[ bucket_ptrs[ key >> shift ]++ ] = key; }
upc_reduced_sum(… ) ;                /* get bucket size totals */
for ( i = 0; i < THREADS; i++ ) {    /* get my bucket from every proc */
    upc_memget(…); }
```
• UPC fine-grain accesses inefficient
• Coarse-grain version closer to MPI (2-3x)
UPC Microbenchmarks

• Compare memory access costs
  – Quantify software overhead

• Private
  – Local memory, local pointer

• Shared-local
  – Local memory, global pointer

• Shared-same-node
  – Non-local memory (but on same SMP node)

• Shared-remote
  – Non-local memory
UPC Microbenchmarks

• Architectures
  – Compaq AlphaServer SC, v1.7 compiler (ORNL)
  – Compaq AlphaServer Marvel, v2.1 compiler (Florida)
  – Sun SunFire 8600 (UMD)
  – AMD Athlon PC cluster (OSU)
  – Cray T3E (MTU)
  – SGI Origin 2000 (UNC)
- Global pointers significantly slower
- Improvement with newer UPC compilers
Observations

• Fine-grain programming model is seductive
  – Fine-grain access to shared data
  – Simple, clean, easy to program

• Not a good reflection of clusters
  – Efficient fine-grain communication not supported in hardware
  – Architectural trend towards clusters, away from Cray T3E
Observations

• Programming model encourages poor performance
  – Easy to write simple fine-grain parallel programs
  – Poor performance on clusters
  – Can code around this, often at the cost of complicating your model or changing your algorithm

• Dubious that compiler techniques will solve this problem
  – Parallel algorithms with block data movement needed for clusters
  – Compilers cannot robustly transform fine-grained code into efficient block parallel algorithms
Observations

• Hybrid programming model is easy to use
  – Fine-grained shared data access easy to program
  – Use coarse-grain message passing for performance
  – Faster code development, prototyping
  – Resulting code cleaner, more maintainable

• Must avoid degrading local computations
  – Allow compiler to fully optimize code
  – Usually not achieved in fine-grain programming
  – Strength of using explicit messages (MPI)
Recommendations

• Irregular coarse-grain algorithms
  – For peak cluster performance, use message passing
  – For quicker development, use hybrid paradigm
    • Use fine-grain remote accesses sparingly
  – Exploit existing code / libraries where possible

• Irregular fine-grain algorithms
  – Execute smaller problems on large SMPs
  – Must develop coarse-grain alternatives for clusters

• Fine-grain programming on clusters still just a dream
  – Though compilers can help for regular access patterns
Impact of Architecture Trends

• **Trends**
  – Faster cluster interconnects (Quadrics, InfiniBand)
  – Larger memories
  – Processor / memory integration
  – Multithreading

• **Raw performance improving**
  – Faster networks (lower latency, higher bandwidth)
  – Absolute performance will improve

• **But same performance limitations!**
  – Avoid small messages
  – Avoid software communication overhead
  – Avoid penalizing local computation
Related Work

• **Parallel paradigms**
  – Many studies
  – PMODELs (DOE / NSF) project

• **UPC benchmarking**
  – T. El-Ghazawi et al. (GWU)
    • Good performance on NAS benchmarks
    • Mostly relies on upc_memput(), upc_memget()
  – K. Yelick et al. (Berkeley)
    • UPC compiler targeting GASNET
    • Compiler attempts to aggregate remote accesses
End of Talk