Berkeley UPC Runtime Report

May 17, 2004

Jason Duell
LBNL
Topics

- Pthreaded runtime
- Support for GCCUPC (Intrepid)
- C++/MPI Interoperability
- Usability/stability improvements:
- Future work
Pthreaded UPC

- Pthreaded version of the runtime
  - Our current strategy for SMPs and clusters of SMPs
- Implementation challenge: thread-local data.
  - Different solution for binary vs. source-to-source
- Has exposed issues in UPC specification:
  - Global variables in C vs. UPC
  - Misc. standard library issues: rand() behavior
Pthreaded UPC

- Future work: implement System V shared memory, and compare to pthreads
  - Benefit: many scientific libraries are not pthread-safe.
  - But: lots of bootstrapping issues, limits on size of shared regions
  - Currently targeting end-of-FY04 for SysV completion

- Pthreads share a single network connection:
  - Fewer network points for fixed number of UPC threads
  - Any pthread can service pending requests for all: better network attentiveness
  - But SysV shared memory may avoid lock contention for network.
Topics

- Pthreaded runtime
- Support for GCCUPC (Intrepid)
- C++/MPI Interoperability
- Usability/stability improvements:
- Future work
GCCUPC (Intrepid) support

- GCCUPC can now use Berkeley UPC runtime
  - Generates binary objects that link with our library.
- GCCUPC previously only for shared memory: now able to use any GASNet network
  - Myrinet, Quadrics, Infiniband, MPI, Ethernet
- Benefits:
  - A network-portable binary UPC compiler now exists for x86, MIPS, future architectures supported by GCCUPC
  - Demonstrates that our runtime can be targeted by a binary compiler (vendors more likely to adopt)
GCCUPC: implementation

• Primary obstacle: inline functions and macros
  - Needed in src-to-src for speed, abstraction layer.
  - But can’t link against them from binary compiler

• Current solution:
  - GCCUPC generates performance-critical logic (ptr manipulation, MYTHREAD, etc.) as binary
  - Convert other inline functions into regular functions
GCCUPC: Future Work

• Support pthreaded executables:
  - Funded, and underway at Intrepid
  - Requires significant changes to GCCUPC’s link and initialization strategy (multiple shared regions, thread-local data support)

• System V Shared Memory support:
  - Should “work out of the box” once runtime supports it

• Add extra inlining pass to GCCUPC:
  - Read our inline function definitions & generate binary code for them
  - Would allow GCCUPC to automatically get our platform-specific shared pointer representations
  - Not funded, but worth funding :)

Topics

- Pthreaded runtime
- Support for GCCUPC (Intrepid)
- C++/MPI Interoperability
- Usability/stability improvements:
- Future work
C++/MPI Interoperability

• Experiment came out of GCCUPC work
  - Needed to publish an explicit initialization API
  - Made sure C++/MPI could use it, so we wouldn’t have to change interface later.

• Motivation: “2nd Front” for UPC acceptance
  - Allow UPC to benefit existing C++/MPI codes
  - Optimize critical sections of code
  - Communication, CPU overlap
  - Easier to implement certain algorithms
  - Easier to use than GASNet
  - Provide transparently in existing libraries (SuperLU)
C++/MPI Interoperability

• **Note:** “This is not UPC++”
  - We’re not supporting C++ constructs within UPC
  - C++/MPI can call UPC functions like regular C functions
  - UPC code can call C functions in C++/MPI code
  - UPC functions can return regular C pointers to local shared data, then convert them back to shared pointers to do communication

• **Status:**
  - Working in both directions: \{C++/MPI\} --&gt; UPC, and vice versa
  - Tested with IBM xIC, Intel ecc, HP cxx, GNU g++, and their MPI versions.
C++/MPI: Future Work

- **Major limitation: can’t share arbitrary data**
  - Can’t share arbitrary global/stack/heap memory: must allocate shared data from UPC calls
  - May require changes to client C++/MPI code, or else use of shared buffers
  - This problem would exist for UPC++, too.

- **Research: allow non-UPC data to be shared**
  - Regular dynamic/heap memory: easy (hijack malloc)
  - Stack/global data: harder (but firehose allows)
  - Would be non-standard UPC extensions
    - May be worth adding to language.
Topics

• Pthreaded runtime
• Support for GCCUPC (Intrepid)
• C++/MPI Interoperability
• Usability/stability improvements:
• Future work
Usability/Stability improvements

• “Brainless” installation for new users
  - Added remote translation over HTTP: low-latency
  - Only need to download/install 5 MB runtime
  - Almost all networks are now autodetected
    - `configure; make; make install`
  - Can install Berkeley UPC in ~5 minutes
  - Over 130 downloads of our 1.1 release
  - Increasing traffic on mailing list and Bugzilla server
Usability/Stability improvements

• Nightly build of runtime on many configurations:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Architecture</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>x86/IA64</td>
<td>GM/MPI</td>
</tr>
<tr>
<td>Tru64</td>
<td>Alpha</td>
<td>Elan/MPI</td>
</tr>
<tr>
<td>AIX</td>
<td>Power 3</td>
<td>LAPI/MPI</td>
</tr>
<tr>
<td>OS X</td>
<td>Power 5</td>
<td>IB/MPI</td>
</tr>
<tr>
<td>SGI Altix</td>
<td>IA64</td>
<td>pthread/MPI</td>
</tr>
<tr>
<td>T3E</td>
<td>Alpha</td>
<td>MPI</td>
</tr>
</tbody>
</table>

• Test suite now contains 250+ test cases
  • works with IBM, Quadrics, PBS batch systems
  • nightly run of test suite on various platforms coming soon
Topics

- Pthreaded runtime
- Support for GCCUPC (Intrepid)
- C++/MPI Interoperability
- Usability/stability improvements:
  - Future work
Future work

• System V shared memory support
• GCCUPC pthreads, inline pass support
• Caching remote shared accesses
  - Toy implementation done as experiment. Saw 100x speedup vs. network for 8 byte gets, but still 50x slower than regular pointer access.
  - Need full implementation and tuning.
    - Architecture/compiler-specific tuning
    - Lookup cost vs. hit rate tradeoff may vary across applications
    - “Smart” runtime cache prediction/prefetching
• Allow regular static/heap data to be shared