Charm++ for Productivity and Performance
A Submission to the 2011 HPC Class II Challenge

Laxmikant V. Kale*
Anshu Arya    Abhinav Bholele    Abhishek Gupta    Nikhil Jain
Pritish Jetley    Jonathan Lifflander    Phil Miller    Yanhua Sun
Ramprasad Venkataraman*    Lukasz Wesolowski    Gengbin Zheng

Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign

*{kale, ramv}@illinois.edu
LLNL-PRES-513271

SC11: November 15, 2011
<table>
<thead>
<tr>
<th>Benchmark Category</th>
<th>Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>Dense LU Factorization</td>
</tr>
<tr>
<td></td>
<td>1D FFT</td>
</tr>
<tr>
<td></td>
<td>Random Access</td>
</tr>
<tr>
<td>Optional</td>
<td>Molecular Dynamics</td>
</tr>
<tr>
<td></td>
<td>Barnes-Hut</td>
</tr>
</tbody>
</table>
Object-based: Express logic via indexed collections of interacting objects (both data and tasks)

Over-decomposed: Expose more parallelism than available processors
Charm++
Programming Model

Runtime-Assisted scheduling, observation-based adaptivity, load balancing, composition, etc.

Message-Driven Trigger computation by invoking remote entry methods

Non-blocking, Asynchronous Implicitly overlapped data transfer

Kale et al. (PPL, Illinois)
Charm++
Program Structure

- Regular C++ code
  - No special compilers
Charm++

Program Structure

- Regular C++ code
  - No special compilers
- Small parallel interface description file
  - Can contain control flow DAG
  - Parsed to generate more C++ code
Charm++

Program Structure

- Regular C++ code
  - No special compilers
- Small parallel interface description file
  - Can contain control flow DAG
  - Parsed to generate more C++ code
- Inherit from framework classes to
  - Communicate with remote objects
  - Serialize objects for transmission
- Exploit modern C++ program design techniques (OO, generics etc)
Charm++

Capabilities

- Promotes natural expression of parallelism
- Supports modularity
Charm++

Capabilities

- Promotes natural expression of parallelism
- Supports modularity
- Overlaps communication and computation
- Automatically balances load
Charm++

Capabilities

- Promotes natural expression of parallelism
- Supports modularity
- Overlaps communication and computation
- Automatically balances load
- Automatically handles heterogenous systems
- Adapts to reduce energy consumption
- Tolerates component failures

For more info
http://charm.cs.illinois.edu/why/
### Metrics: Performance

#### Our Implementations in Charm++

<table>
<thead>
<tr>
<th>Code</th>
<th>Machine</th>
<th>Max Cores</th>
<th>Best Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU</td>
<td>Cray XT5</td>
<td>8K</td>
<td>67.4% of peak</td>
</tr>
<tr>
<td>FFT</td>
<td>IBM BG/P</td>
<td>64K</td>
<td>2.512 TFlop/s</td>
</tr>
<tr>
<td>RandomAccess</td>
<td>IBM BG/P</td>
<td>64K</td>
<td>22.19 GUPS</td>
</tr>
<tr>
<td>MD</td>
<td>Cray XE6</td>
<td>16K</td>
<td>1.9 ms/step (125K atoms)</td>
</tr>
<tr>
<td>Barnes-Hut</td>
<td>IBM BG/P</td>
<td>16K</td>
<td>11.6 ms/step (1M atoms)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>27 × 10⁹ interactions/s</td>
</tr>
</tbody>
</table>
## Metrics: Code Size

### Our Implementations in Charm++

<table>
<thead>
<tr>
<th>Code</th>
<th>C++</th>
<th>CI</th>
<th>Total¹</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU</td>
<td>1231</td>
<td>418</td>
<td>1649</td>
<td>BLAS</td>
</tr>
<tr>
<td>FFT</td>
<td>112</td>
<td>47</td>
<td>159</td>
<td>FFTW, Mesh</td>
</tr>
<tr>
<td>RandomAccess</td>
<td>155</td>
<td>23</td>
<td>178</td>
<td>Mesh</td>
</tr>
<tr>
<td>MD</td>
<td>645</td>
<td>128</td>
<td>773</td>
<td></td>
</tr>
<tr>
<td>Barnes-Hut</td>
<td>2871</td>
<td>56</td>
<td>2927</td>
<td>TIPSY</td>
</tr>
</tbody>
</table>

**C++** Regular C++ code  

**CI** Parallel interface descriptions and control flow DAG

¹Required logic, excluding test harness, input generation, verification, etc.

Remember: Lots of freebies!

- automatic load balancing, fault tolerance, overlap, composition, portability
## Metrics: Code Size

Our Implementations in Charm++

<table>
<thead>
<tr>
<th>Code</th>
<th>C++</th>
<th>CI</th>
<th>Total¹</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU</td>
<td>1231</td>
<td>418</td>
<td>1649</td>
<td>BLAS</td>
</tr>
<tr>
<td>FFT</td>
<td>112</td>
<td>47</td>
<td>159</td>
<td>FFTW, Mesh</td>
</tr>
<tr>
<td>RandomAccess</td>
<td>155</td>
<td>23</td>
<td>178</td>
<td>Mesh</td>
</tr>
<tr>
<td>MD</td>
<td>645</td>
<td>128</td>
<td>773</td>
<td></td>
</tr>
<tr>
<td>Barnes-Hut</td>
<td>2871</td>
<td>56</td>
<td>2927</td>
<td>TIPSY</td>
</tr>
</tbody>
</table>

**C++** Regular C++ code

**CI** Parallel interface descriptions and control flow DAG

---

Remember: Lots of freebies!

automatic load balancing, fault tolerance, overlap, composition, portability

---

¹Required logic, excluding test harness, input generation, verification, etc.

Kale et al. (PPL, Illinois)  
Charm++ for Productivity and Performance  
SC11: November 15, 2011  
8 / 25
LU: Capabilities

- Composable library
  - Modular program structure
  - Seamless execution structure (interleaved modules)
LU: Capabilities

- Composable library
  - Modular program structure
  - Seamless execution structure (interleaved modules)

- Block-centric
  - Algorithm from a block’s perspective
  - Agnostic of processor-level considerations
LU: Capabilities

- Composable library
  - Modular program structure
  - Seamless execution structure (interleaved modules)

- Block-centric
  - Algorithm from a block’s perspective
  - Agnostic of processor-level considerations

- Separation of concerns
  - Domain specialist codes algorithm
  - Systems specialist codes tuning, resource mgmt etc

<table>
<thead>
<tr>
<th>Lines of Code</th>
<th>Module-specific Commits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CI</td>
</tr>
<tr>
<td>Factorization</td>
<td>517</td>
</tr>
<tr>
<td>Mem. Aware Sched.</td>
<td>9</td>
</tr>
<tr>
<td>Mapping</td>
<td>10</td>
</tr>
</tbody>
</table>
LU: Capabilities

- Flexible data placement
  - Experiment with data layout
- Memory-constrained adaptive lookahead
LU: Performance

Weak Scaling: (N such that matrix fills 75% memory)

![Graph showing weak scaling performance on XT5. The graph plots Total TFlop/s on the y-axis against Number of Cores on the x-axis. The theoretical peak and weak scaling lines are shown, with key performance percentages marked at each data point. The percentages shown are 67%, 67.1%, 67.4%, 66.2%, 67.4%, and 65.7% at core counts of 128, 1024, 8192, and 8192, respectively.]
LU: Performance

... and strong scaling too! (N=96,000)
FFT: Parallel Coordination Code

doFFT()

for(phase = 0; phase < 3; ++phase) {
    atomic {
        sendTranspose();
    }
    for(count = 0; count < P; ++count)
        when recvTranspose[phase] (fftMsg *msg) atomic {
            applyTranspose(msg);
        }
    if (phase < 2) atomic {
        fftw_execute(plan);
        if(phase == 0)
            twiddle();
    }
}
FFT: Performance
IBM Blue Gene/P (Intrepid), 25% memory, ESSL /w fftw wrappers

Cores
GFlop/s

P2P All-to-all
Mesh All-to-all
Serial FFT limit

Kale et al. (PPL, Illinois)
Charm++ for Productivity and Performance
FFT: Performance
IBM Blue Gene/P (Intrepid), 25% memory, ESSL /w fftw wrappers

Charm++ all-to-all
Asynchronous, Non-blocking, Topology-aware, Combining, Streaming

Charm++ for Productivity and Performance
Random Access

What Charm++ brings to the table

Productivity
- Automatically detect completion by sensing quiescence
- Automatically detect network topology of partition

Performance
- Uses same Charm++ all-to-all
Random Access: Performance
IBM Blue Gene/P (Intrepid), 2 GB of memory per node

GUPS vs. Number of cores for Perfect Scaling and Charm++
Kale et al. (PPL, Illinois)
Optional Benchmarks

Why MD and Barnes-Hut?

- Relevant scientific computing kernels
- Challenge the parallelization paradigm
  - Load imbalances
  - Dynamic communication structure
- Express non-trivial parallel control flow
Molecular Dynamics
Overview

1. Mimics force calculation in NAMD
2. Resembles the miniMD application in the Mantevo benchmark suite
3. SLOC is 773 in comparison to just under 3000 lines for miniMD

(a) 1 Away Decomposition  (b) 2 AwayX Decomposition
MD: Performance
125,000 atoms. Cray XE6 (Hopper)

Performance on Hopper (125,000 atoms)

Time per step (ms)
Number of cores
Performance on Hopper (125,000 atoms)
1.91 ms/step
No LB
Refine LB

Kale et al. (PPL, Illinois)
Charm++ for Productivity and Performance
MD: Performance

1 million atoms. IBM Blue Gene/P (Intrepid)

Speedup on Intrepid (1 million atoms)

Ideal
Charm++

11.6 ms/step
MD: Performance

Number of cores does **not** have to be a power-of-2
Adaptive overlap of computation and communication allows latency of requests for remote data to be hidden by useful local computation on PEs.

Automatic measurement-based load balancing allows dissociation of data decomposition from task assignment: balance communication through Oct-decomposition and computation through separate load balancing strategy.
Barnes-Hut: Performance

Non-uniform (Plummer) distribution. IBM Blue Gene/P (Intrepid)

Barnes-Hut scaling on BG/P

Kale et al. (PPL, Illinois)  Charm++ for Productivity and Performance
Barnes-Hut: Performance
Non-uniform (Plummer) distribution. IBM Blue Gene/P (Intrepid)

Barnes-Hut scaling on BG/P

Plummer 100k Distribution

Distance from COM

Frequency

Time/step (seconds)

Cores

2k 4k 8k 16k

50m
10m

Kale et al. (PPL, Illinois)
Temperature-aware load balancing  Tue @ 2:00 pm
Fault tolerance protocol  PhD Forum; Tue @ 3:45 pm
NAMD at 200K+ cores  Thu @ 11:00 am
Topology aware mapping for PERCS  Thu @ 4:00 pm
Parallel stochastic optimization  Poster
All-to-all simulations on PERCS  Poster

For more info
http://charm.cs.illinois.edu/why/
MeshStreamer: Message Routing and Aggregation