Charm++
Migratable Objects + Asynchronous Methods + Adaptive Runtime
= Performance + Productivity

Laxmikant V. Kale*
Anshu Arya       Nikhil Jain      Akhil Langer      Jonathan Lifflander
Harshitha Menon  Phil Miller      Xiang Ni         Yanhua Sun
Ehsan Totoni     Ramprasad Venkataraman*  Lukasz Wesolowski

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

*{kale, ramv}@illinois.edu
## Metrics: Performance and Productivity

Our Implementations in Charm++

<table>
<thead>
<tr>
<th>Productivity</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Benchmark</td>
</tr>
<tr>
<td></td>
<td>CI</td>
</tr>
<tr>
<td>C++</td>
<td></td>
</tr>
<tr>
<td>CI</td>
<td></td>
</tr>
</tbody>
</table>

### Required Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Code</th>
<th>CI</th>
<th>Benchmark</th>
<th>Driver</th>
<th>Total</th>
<th>Machine</th>
<th>Max</th>
<th>Performance Highlight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D FFT</td>
<td>54</td>
<td>29</td>
<td>83</td>
<td>102</td>
<td>185</td>
<td>BG/P</td>
<td>64K</td>
<td>2.71 TFlop/s 2.31 TFlop/s</td>
</tr>
<tr>
<td>Random Access</td>
<td>76</td>
<td>15</td>
<td>91</td>
<td>47</td>
<td>138</td>
<td>BG/P</td>
<td>128K</td>
<td>43.10 GUPS 15.00 GUPS</td>
</tr>
<tr>
<td>Dense LU</td>
<td>1001</td>
<td>316</td>
<td>1317</td>
<td>453</td>
<td>1770</td>
<td>XT5</td>
<td>8K</td>
<td>55.1 TFlop/s (65.7% peak)</td>
</tr>
</tbody>
</table>

### Additional Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Code</th>
<th>CI</th>
<th>Benchmark</th>
<th>Driver</th>
<th>Total</th>
<th>Machine</th>
<th>Max</th>
<th>Performance Highlight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Molecular Dynamics</td>
<td>571</td>
<td>122</td>
<td>693</td>
<td>n/a</td>
<td>693</td>
<td>BG/P</td>
<td>128K</td>
<td>24 ms/step (2.8M atoms) 44 ms/step (2.8M atoms)</td>
</tr>
<tr>
<td>AMR</td>
<td>1126</td>
<td>118</td>
<td>1244</td>
<td>n/a</td>
<td>1244</td>
<td>BG/Q</td>
<td>32k</td>
<td>22 steps/sec, 2d mesh, max 15 levels refinement</td>
</tr>
<tr>
<td>Triangular Solver</td>
<td>642</td>
<td>50</td>
<td>692</td>
<td>56</td>
<td>748</td>
<td>BG/P</td>
<td>512</td>
<td>48x speedup on 64 cores with helm2d03 matrix</td>
</tr>
</tbody>
</table>

- **C++** Regular C++ code
- **CI** Parallel interface descriptions and control flow DAG
Metrics++: Differentiating Capabilities
Demonstrated Productivity Benefits

This year’s submission demonstrates:

- Interoperating with MPI
- Asynchronous, non-blocking collectives
- Dynamic load balancing with closed loop control system
- Automatic checkpoints for split execution
- Tolerating process failures
This year’s submission demonstrates:

- **FFT**  Interoperating with MPI

- **FFT, RA, LU**  Asynchronous, non-blocking collectives

- **MD, AMR**  Dynamic load balancing with closed loop control system

- **MD**  Automatic checkpoints for split execution

- **MD, AMR**  Tolerating process failures
Metrics++: Differentiating Capabilities
Demonstrated Productivity Benefits

This year’s submission demonstrates:

**FFT** Interoperating with MPI
- gradual, incremental adoption path

**FFT, RA, LU** Asynchronous, non-blocking collectives
- trivially overlaps collectives with computation

**MD, AMR** Dynamic load balancing with closed loop control system
- automates “when” and “how” decisions

**MD** Automatic checkpoints for split execution
- checkpoint on $x$ cores, automatic restart on $y$ cores

**MD, AMR** Tolerating process failures
- continues execution despite randomly injected failures
Charm++ Programming Model

Migratable Objects
- Globally visible
- Migratable
- Overdecomposed
- Data / Task

Async Methods
- Invoke remotely
- Non-blocking
- Non-preemptive
- Prioritized

Adaptive Runtime
- Message-driven
- Orchestrates
- Observes
- Adapts

Ensuing Benefits
overlap, load balance, adaptivity, fault tolerance, elegance, modularity, composability ...
Required Benchmarks

- FFT
- Random Access
- Dense LU Factorization
for(phase = 0; phase < 3; ++phase) {
    serial { initStreamer(streamer); }
    when streamerReady() serial {
        sendTranspose(...);
    }
    for(count = 0; count < P; ++count)
        when recvData[phase] (...) serial {
            applyTranspose(data, n, src);
        }
    if (phase < 2) serial {
        fftw_execute(plan);
        if(phase == 0)
            twiddle();
    }
}
1D FFT
IBM BG/P (Intrepid), BG/Q (Vesta), 25% memory, ESSL /w fftw wrappers

Kale et al. (PPL, Illinois)
Charm++
SC12: November 13, 2012
Demonstrated Capability: Interoperability

Invoke Charm++ modules from MPI

- Callable like any external MPI library
- Incremental adoption
- Mix-and-match models to needs

![Diagram](image)

(a) Time Sharing
(b) Space Sharing
(c) Combined

Kale et al. (PPL, Illinois)
Expressed in < 100 SLOC as point-to-point sends, with optimizations left to the Charm++ TRAM library
Demonstrated Capability: Non-blocking Collectives
Asynchronous, Non-blocking, Topology-aware, Combining, Streaming Many-to-many

- Use point to point sends and let Charm++ optimize communication
- Automatically detect and adapt to network topology of partition
- Topological Routing and Aggregation Module (TRAM)
  - Aggregation of fine-grained communication (Random Access)
  - Minimal topology-aware software routing with recombining at intermediate destinations
  - Streaming to lower memory use and improve network bandwidth utilization
  - Performance improved up to 10x for Random Access and FFT
Complete parallel control flow expressed in 306 lines!
Complete parallel control flow expressed in 306 lines!

- Block-centric
  - Algorithm from a block’s perspective
  - Agnostic of processor-level considerations
- Memory-constrained adaptive lookahead
- Flexible data placement

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

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

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

... and strong scaling too! (N=96,000)
1. Mimics short-range force calculation in NAMD
2. Similar to Mantevo’s *miniMD* \((SLOC \approx 3000)\), but in \(< 700\) SLOC

Hybrid force-spatial decomposition
Demonstrated Capability: Automatic, Load Balancing

How to...

Call `AtSync()` in code every iteration
Pass `+MetaLB` on command line

- Too frequent load balancing increases total execution time
- Infrequent load balancing leads to load imbalance
- Meta-Balancer adaptively performs load balancing to obtain best total execution time

### Elapsed time vs LB Period (BlueGene/P)

<table>
<thead>
<tr>
<th>Cores</th>
<th>Periodic (s)</th>
<th>MetaLB (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8k</td>
<td>504</td>
<td>413</td>
</tr>
<tr>
<td>16k</td>
<td>260</td>
<td>277</td>
</tr>
<tr>
<td>32k</td>
<td>131</td>
<td>131</td>
</tr>
<tr>
<td>64k</td>
<td>104</td>
<td>100</td>
</tr>
<tr>
<td>128k</td>
<td>54</td>
<td>52</td>
</tr>
</tbody>
</table>

Kale et al. (PPL, Illinois)
LeanMD
2.8 million atoms. IBM BG/P (Intrepid)

Performance on Intrepid (2.8 million atoms)

Time per step (ms)
Number of cores
Performance on Intrepid (2.8 million atoms)
No LB
Periodic LB
Meta LB

Kale et al. (PPL, Illinois)
Demonstrated Capability: Tolerating a Failed Process
Checkpoint and Restart Time

How to...
Call CkStartMemCheckpoint() at desired checkpoint interval

LeanMD Checkpoint Time on BlueGene/Q
![Graph showing LeanMD Checkpoint Time on BlueGene/Q]

LeanMD Restart Time on BlueGene/Q
![Graph showing LeanMD Restart Time on BlueGene/Q]
Adaptive Mesh Refinement (sloc: 1244)

Sample simulation

Propagation of refinement decision messages
Adaptive Mesh Refinement

- Block-centric
  - Algorithm from a block’s perspective
  - Agnostic of processor-level considerations
- Easily address a mesh-block with bit-vector indices
  - Charm++ handles physical locations
- Dynamic, distributed load balancing

Algorithmic Improvements\(^1\)

- \(O(\#\text{blocks}/P)\) vs \(O(\#\text{blocks})\) memory per process
- 2 system quiescence states vs \#level reductions for mesh restructuring
- \(O(1)\) vs \(O(\log P)\) time neighbor lookup

---

Adaptive Mesh Refinement

Timesteps per second strong scaling on IBM BG/Q with a max depth of 15.

Kale et al. (PPL, Illinois)

SC12: November 13, 2012 20 / 26
Sparse Triangular Solver (sloc: 692)

Matrix Decomposition

\[
\begin{bmatrix}
l_{11} & l_{21} & l_{22} \\
l_{33} & l_{44} & l_{55} \\
l_{43} & l_{54} & l_{55} \\
l_{81} & & \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
x_8 \\
x_9 \\
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5 \\
b_6 \\
b_7 \\
b_8 \\
b_9 \\
\end{bmatrix}
\]

Column Decomposition

Multicast

Dense Parts Decomposition

Kale et al. (PPL, Illinois)

Charm++
Sparse Triangular Solver
Productivity in Charm++

- Methods express dependencies
  - Avoid MPI_Iprobe, MPI_Test etc

- Overdecomposition and round-robin mapping
  - overlap and load balance

- Dynamic creation of parallel units
  - Distributes dense matrix regions

- SLOC (692 vs 897 for SuperLU)
  - more sophisticated, higher performance algorithm
Sparse Triangular Solver
Parallel Algorithm

Schedule independent computation

High priority: Process messages

Low priority: do independent computation

Highest priority: receive solution values

High priority: Process messages

if (onDiagonalChare) {
    serial { thisProxy[thisIndex].indepCompute(...) }
    overlap {
        while (!finished) {
            when recvData(int len, double data[len], int rows[len])
                serial { if(len>0) diagReceiveData(len, data, rows); }
        }
        when indepCompute(int a) serial { myIndepCompute(); }
    } else {
        when getXvals(xValMsg* msg) serial { nondiag_compute(); }
        while (!finished) {
            when recvData(int len, double data[len], int rows[len])
                serial { nondiagReceiveData(len, data, rows); }
        }
    }
}
Sparse Triangular Solver
Performance

Number of Cores
slu_webbase-1M
slu_helm2d03
slu_hood
slu_largebasis
SuperLU_largebasis
SuperLU_webbase-1M
SuperLU_helm2d03
SuperLU_hood

Solution Time (s)

640µs

Kale et al. (PPL, Illinois)
Charm++
SC12: November 13, 2012
Charm++ and PPL at SC12

Charm++ Tutorial 8:30AM-12:00PM on Sunday November 11 (past)

HPC Challenge BoF
12:15PM-1:15PM on Tuesday November 13, in 255-A
(you’re here!)

Fernbach Award Talk
11:30AM-12:00PM on Wednesday November 14th, in 155-E

Doctoral Showcase 11:15AM - 11:30AM on Wednesday, in 155-F, by Osman Sarood ”Saving Energy and Power”

Paper Presentation 2:00-2:30PM on Wednesday Nov 14, in 355-EF, Talk by Yanhua Sun ”Optimizing fine-grained communication in a biomolecular dynamics simulation application on Cray XK6”

Charm++ BoF 12:15PM-1:15PM on Thursday November 15, in 255-A

For more info
http://charm.cs.illinois.edu/
LeanMD: Adding FT and LB

Trivial additions but major gains

```cpp
serial { AtSync(); }
when ResumeFromSync() { }
```
LeanMD: Adding FT and LB
Trivial additions but major gains

```
serial { AtSync(); }
when ResumeFromSync() { }

if (stepCount % checkptFreq == 0) {
    serial {
        //coordinate to start checkpointing
        contribute(CkCallback(CkReductionTarget(Cell,startCheckpoint),thisProxy(0,0,0)));
    }
    if (thisIndex.x == 0 && thisIndex.y == 0 && thisIndex.z == 0) {
        when startCheckpoint() serial {
            CkCallback cb(CkReductionTarget(Cell,recvCheckPointDone), thisProxy);
            if (checkptStrategy == 0) CkStartCheckpoint(logs.c_str(), cb);
            else CkStartMemCheckpoint(cb);
        }
    }
    when recvCheckPointDone() { }
}
```
LeanMD: Adding FT and LB
Trivial additions but major gains

```c
serial { AtSync(); }
when ResumeFromSync() {} 

if (stepCount % checkptFreq == 0) {
    serial {
        //coordinate to start checkpointing
        contribute(CkCallback(CkReductionTarget(Cell,startCheckpoint),thisProxy(0,0,0)));
    }
    if (thisIndex.x == 0 && thisIndex.y == 0 && thisIndex.z == 0) {
        when startCheckpoint() serial {
            CkCallback cb(CkReductionTarget(Cell,recvCheckPointDone), thisProxy);
            if (checkptStrategy == 0) CkStartCheckpoint(logs.c_str(), cb);
            else CkStartMemCheckpoint(cb);
        }
        when recvCheckPointDone() {}
    }
}

//kill one of processes to demonstrate fault tolerance
if (stepCount == 30 && thisIndex.x == 1 && thisIndex.y == 1 && thisIndex.z == 0) serial {
    if (CkHasCheckpoints()) {
        CkDieNow();
    }
}
```