Flexible Hierarchical Execution of Parallel Task Loops

Michael Robson, Villanova University
Kavitha Chandrasekar, University of Illinois Urbana-Champaign
Injection Bandwidth vs CPU speeds

Bytes to Flops ratio

Gflops/node

XT3 △ BG/P
△ XT4
BG/L △ XT5 △ BG/Q
△ XE6
△ XC30
Phi (Stampede)

XK7 (Titan)

Sierra*

Summit*
Motivation

• Trend:
  • Deeper nodes
  • Thinner pipes

• Accelerators (e.g. GPUs)

• Increased Programmer effort

<table>
<thead>
<tr>
<th>Year</th>
<th>Machine</th>
<th>Linpack (FLOPs)</th>
<th>FLOPs/Local</th>
<th>FLOPs/Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td>1988</td>
<td>Cray YMP</td>
<td>2.1 Giga</td>
<td>0.52</td>
<td>0.52</td>
</tr>
<tr>
<td>1997</td>
<td>ASCI Red</td>
<td>1.6 Tera</td>
<td>8.3</td>
<td>20</td>
</tr>
<tr>
<td>2011</td>
<td>Road-runner</td>
<td>1.0 Peta</td>
<td>6.7</td>
<td>170</td>
</tr>
<tr>
<td>2012</td>
<td>Sequoia</td>
<td>17 Peta</td>
<td>32</td>
<td>160</td>
</tr>
<tr>
<td>2013</td>
<td>Titan</td>
<td>18 Peta</td>
<td>29</td>
<td>490</td>
</tr>
<tr>
<td>2018</td>
<td>Summit</td>
<td>122 Peta</td>
<td>37</td>
<td>1060</td>
</tr>
<tr>
<td>2011</td>
<td>K-Comp</td>
<td>11 Peta</td>
<td>15</td>
<td>95</td>
</tr>
<tr>
<td>2013</td>
<td>Tianhe-2</td>
<td>34 Peta</td>
<td>22</td>
<td>1500</td>
</tr>
<tr>
<td>2016</td>
<td>Sunway</td>
<td>93 Peta</td>
<td>130</td>
<td>1500</td>
</tr>
<tr>
<td>2021</td>
<td>TBD</td>
<td>1.0 Exa</td>
<td>80</td>
<td>3200</td>
</tr>
<tr>
<td>2021</td>
<td>TBD</td>
<td>1.0 Exa</td>
<td>300</td>
<td>10000</td>
</tr>
</tbody>
</table>

S. Plimpton (Charm ’19)
Fat Nodes

First law of holes:

• If you find yourself in a hole, stop digging!

We are digging ourselves deeper into a node
Main Idea: *Spreading* Work Across Cores

• Speed up individual calculations via OpenMP
• FLOPs are cheap, need to inject early
• Better communication, computation overlap
Motivation

New Axes of Optimization

• Problem Size Decomposition (Grain Size)
• Resources Assigned to a Task (Spreading)
Experimental Setup

• Charm Build
  • **Separate processes** (Non-SMP mode)
  • `-O3` –with-production
  • PAMI-LRTS communication layer

• Five Runs
  • OpenMP Threads (Spreading) = 1, 2, etc
  • Grid Size = $178848^2$ doubles (~90%)
  • Block Size = 7452, various
  • Chares (Objects) = $24^2$
  • Iterations = 10-100
  • Nodes = 4
OpenMP Pragmas

• Schedule - Static
• Chunk Size (Iterations)
  • Default (Block / Cores)
    • 1
    • 16
    • 512
• Collapse
Machines

Bridges (PSC)
- 2 x 14-core Haswell E5-2695
- 128 GB DDR4

Summit (ORNL)
- 2 x 22-core IBM Power9
- 512 GB DDR4
Bridges

The bar chart illustrates the time taken for various types of bridges under different thread counts.

- Static (D)
- Static (1)
- Collapse + Static (D)
- Charm++

The x-axis represents the type of bridge, and the y-axis represents time. Different thread counts are indicated by different colors:
- 1
- 2
- 3
- 4
- 6
- 8
- 12
- 24

The chart shows that the time taken increases with the number of threads for all types of bridges, but the increase varies depending on the type of bridge and the specific configuration.
Summit – Block Size

![Graph showing block size comparison across different types. The x-axis represents Block Size, and the y-axis represents Time. The graph shows data for Charm(1), Spreading(2), Spreading(3), Spreading(6), and Spreading(7). The bars indicate the performance metrics at various block sizes, with the highest performance at Block Size 621, followed by 7452, 14904, 19872, 22356, and 29808.](image-url)
Summit – Scaling
What happens when we eliminate communication?

i.e. are effects just from improved caching?
Summit – No Send

![Bar chart showing performance metrics for different block sizes and thread types. The chart compares Charm++, Spreading, and OpenMP configurations.](chart.png)
Let's look at communication performance...

using projections.
OpenMP Baseline

Received bytes per second

Time (s)

320K
240K
160K
80K
0
17.5 21.9 26.3 30.7 35.1
Charm++ Baseline

Received bytes per second

Time (s)

6.8 11.2 15.6 20.0 24.4
Spreading Technique

![Graph showing received bytes per second over time](image)

- Received bytes per second:
  - 320K
  - 240K
  - 160K
  - 80K
  - 0

- Time (s):
  - 22.4
  - 26.8
  - 31.2
  - 35.6
  - 40.0
Time (s)

Received bytes per second

0 80K 160K 240K 320K

6.8 11.2 15.6 20.0 24.4
Runtime Integration
Automating teams configuration

• Broader Agenda
  • Automate decisions -> easier for user
  • “Spread”: How many teams, i.e how many masters and how many drones?
  • Other runtime decisions:
    • How many ppn, i.e cores per process?
    • How many processes per node?
    • How many cores to turn off (memory bottleneck)?
    • Enable SMT or not?
Automating teams configuration

• Use OpenMP to create master thread on all cores
• Integrate with load balancing framework to change master thread count
• Use OpenMP nested parallelism to set/change number of drone threads within the application
  • Use pthread affinities instead of OpenMP affinity to update configurations at runtime
• Runtime selects the best performing configuration after testing with different configurations (one per LB step)
Using OpenMP with nested parallelism (static)

Bridges - single-node integrated OpenMP runs for SMP and Non-SMP builds
Using OpenMP with nested parallelism (static)

Stampede2 - Skylake 4-node run integrated OpenMP

Execution on 4 nodes (Stampede2 Skx)
Charm++ SMP + OpenMP

Execution Time (seconds) vs Num PEs

Collapse + Static
Static
OpenMP Implementation

machine-smp.C

```c
int num_threads = tocreate + 1;
omp_set_dynamic(0);
omp_set_num_threads(num_threads);
#pragma omp parallel
{
    size_t i = omp_get_thread_num();
call_startfn((void *)i);
}
```

jacobi2d.C

```c
#pragma omp parallel for private(temperatureIth, \
            difference) num_threads(drones_per_pe)
for(int i=istart; i<ifinish; ++i) {
    for(int j=jstart; j<jfinish; ++j) {
        temperatureIth=(temperature[i][j]
        + temperature[i-1][j]
        + temperature[i+1][j]
        + temperature[i][j-1]
        + temperature[i][j+1]) * 0.2;
```

Static configuration:
- OMP_NESTED=true
- GOMP_CPU_AFFINITY=0-7 (eg, for 8 cores)
- OMP_PROC_BIND=spread

Dynamic configuration:
- pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
OpenMP implementation with pthread affinity

- Similar performance with process-based and OpenMP implementations
  - Some NUMA effects
- OpenMP Limitations:
  - Nested parallelism configurations cannot be dynamically changed
  - Affinities are set at the initialization and cannot be changed
- With Charm++ we are able to dynamically change OpenMP configurations and with pthread affinity we set affinities for each new configuration
Next steps

• Integrate the LB framework to fully automate configuration selection
  • Current implementation is able to dynamically set different configurations at runtime based on user input
  • Benefit over static OpenMP configuration – configurations and affinities can be changed at runtime

• Compare with CkLoop implementation in Charm++
Summary

• Spreading offers new optimization parameter
• Increases performance 20-30% in prototype application
• Spread factor is controllable at runtime
• Current integration into Charm++ ongoing

Questions

Michael Robson michael.robson@villanova.edu
Kavitha Chandrasekar kchndrs2@illinois.edu