The High Performance Cluster for Lattice QCD Calculations:
System Monitoring and Benchmarking

Part II – BENCHMARKING

Michal Kapalka
kapalka@icslab.agh.edu.pl
Summer student @ DESY Hamburg

Supervisor: Andreas Gellrich

September 2002
Outline

- BENCHMARKS – AN INTRODUCTION
- SINGLE-NODE BENCHMARKS
- PARALLEL COMPUTING & MPI
- HOW TO BENCHMARK A CLUSTER?
  - POINT-TO-POINT COMMUNICATION
  - COLLECTIVE COMMUNICATION
- SUMMARY & CONCLUSIONS & QUESTIONS
Benchmarks – WHY?

- Comparing – different hardware/software
- Testing a given configuration & finding bottlenecks

- Relative simplicity
- Comparing
- Testing

- That's what we're going to talk about…
WHAT to test?

- Single machine
  - CPU + memory + …

- Cluster or parallel computer
  - communication:
    - interprocessor
    - inter-node

→  

→  

→  


HOWTO part I – one node

- Lattice QCD basic operations: Dirac operator, complex matrices, square norm, ...

- QCD Benchmark (Martin Lüscher)

- Optimization: SSE (PIII), SSE2 (P4) – operations on 2 doubles at once, cache prefetching (PIII) -> www.intel.com
# QCD Benchmark – results

<table>
<thead>
<tr>
<th></th>
<th>D_psi [Mflops]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Xeon 2 GHz</strong></td>
<td>92186127554</td>
</tr>
<tr>
<td><strong>Xeon 1.7 GHz</strong></td>
<td>13468822160</td>
</tr>
<tr>
<td><strong>PIII 800 MHz</strong></td>
<td>67839999999</td>
</tr>
</tbody>
</table>
QCD Benchmark – results (2)

<table>
<thead>
<tr>
<th>Node</th>
<th>Model</th>
<th>GHz</th>
<th>Cache Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>node10</td>
<td>Xeon 2</td>
<td>2</td>
<td>512 KB</td>
</tr>
<tr>
<td>node20</td>
<td>Xeon 1.7</td>
<td>1.7</td>
<td>256 KB</td>
</tr>
<tr>
<td>pal01</td>
<td>PIII 800</td>
<td>0.8</td>
<td>256 KB</td>
</tr>
</tbody>
</table>

- 64 bit: SSE
- 32 bit: SSE(2)

Add assign field
\[ \psi(k) = \psi(k) + c \psi(l) \]
HOWTO part II – a cluster

- CPUs & nodes have to communicate

- CPUs: shared memory

- Nodes: sockets (grrrr…), virtual shared memory (hmm…), PVM, MPI, etc.

- For clusters: MPI (here: MPICH-GM)

  → that's exactly what I've tested

- Remark: communication OVERHEAD
MPI – point-to-point

- Calls: blocking & non-blocking
  - (init + complete)
  - send, receive
  - init x 2, complete x 2
  - Computation...

- Modes: standard, synchronous, buffered, ready

- Uni-or bidirectional?

- Basic operations: send and receive
First step – POE

- Extremely simple – ping-pong test
- Only blocking, standard-mode communication
- Not user-friendly
- But…
POE – results

Point-to-point communication
interprocessor vs. inter-node

- Non-local
  - details later

- Local, no shmem
  - slow (90 MB/s)

- Local with shmem
  - fast (esp. 31-130 KB), but…

DESY Summer Student Programme 2002
Michal Kapalka
My point-to-point benchmarks

- Using different MPI modes: standard, synchronous & buffered (no ready-mode)
- Blocking & non-blocking calls
- Fully configurable via command-line options
- Text and LaTeX output
- But still ping-pong tests
Problems...

- Time measuring
  - CPU time → seems natural, but very low resolution on Linux (clock() call)
  - Real time → high resolution, but can be misleading on overloaded nodes (gettimeofday() call)

- MPICH -GM bug – problems when using shared memory
Results (1)

Point-to-point communication
send and receive in standard mode

- Peak: 1575 MB/s, drops to 151 MB/s @ 16 KB
- Total: max 151 MB/s
Results (2)

- Send & rcv completely different.
- Losing sync.
- Send: peak 1205 MB/s.
- Total: max 120 MB/s.

Point-to-point communication send and receive in buffered mode.

Bandwidth [MB/s]

Message size [B]
Results (3)

Point-to-point communication
blocking calls

- Total bandwidth!
- Standard, seems to be the fastest
- Buffered, use with care
Results (4)

- Blocking: max 151 MB/s
- Non-blocking: max 176 MB/s
- + computation

WHY???
when is it bidirectional?
Uni- or bidirectional?

**Blocking** communication:

Node A: send \rightarrow receive

Node B: receive \rightarrow send

**Non-blocking** communication:

Node A: init x 2 \rightarrow complete

Node B: init x 2 \rightarrow complete

MSG

time
Results (5)

- Non-blocking calls use full duplex
- Also MPI_Sendrecv
- Blocking calls cannot use it, that’s why they’re slower
Results (last but not least)

- The 'blocking calls story' repeats...
- However, buffered mode can be sometimes the most efficient
Point-to-point – conclusions

- Use standard-mode, non-blocking communication whenever it's possible
- Use large messages

1. Write your parallel program
2. Benchmark
3. Analyze
4. Improve
5. Go to 2
Collective communication

- Collective operations:
  - Broadcast
  - Gather, gather to all
  - Scatter
  - All to all gather/scatter
  - Global reduction operator, all reduce

- Root and non-root nodes

- Can be implemented with point-to-point calls, but this can be less effective
What to measure?

\[ b = \frac{N \cdot M}{t} \]

\[ b_{\text{summary}} = b \cdot (K - 1) \]
Results – example #1

Collective communication

Summary bandwidth [MB/s]

Message size [B]

Root: max 527 MB/s, drops down @ 16 KB

Non-root: max 229 MB/s

Saturation: 227 MB/s
Results – example #2

- Very effective algorithm used
- Max: around 400 MB/s

Collective communication

- 4 nodes
- 12 nodes

Summary bandwidth [MB/s]

Message size [B]
Results – example #3

Collective communication

gather to all

- Saturation: 75 MB/s
- BUT…
But…

But…

\[ b_{\text{summary}} = \frac{N \cdot M \cdot (K - 1)}{t} \]

So we should write:

\[ b_{\text{summary}} = \frac{N \cdot M \cdot (K - 1) \cdot K}{t} \]

This needs to be changed
Results – example #4

- Max: 960 MB/s for 12 nodes (160 MB/s per connection)
- Hard job to improve that
Results – example #n

Collective communication
global reduce operator (sum)

Strange behaviour
Stable for message size > 16 KB (max 162 MB/s)

Interpretation → very difficult
Collective – conclusions

- Collective communication is usually not used too often, so one doesn't need to improve its speed.

- However, if it's a must, in some cases, changing collective to point-to-point in a smart way can improve things a little. Also playing with message sizes can help a lot, but be careful.
To do...

- Bidirectional communication
- More flexible method for computing summary bandwidth in collective communication
- Some other benchmarks – closer to the lattice QCD computations
- And the most important – parallelizing all the lattice QCD programs and making use of the benchmarks & results
Summary

- CPU benchmarks can speed up serial programs (running on one node)
- For parallel computations, the real bottleneck is communication, and this has to be tested carefully.
- The interpretation of the results is NOT as important as using them to tune a program and make it fast.