Linux Performance on IBM System z Enterprise

Christian Ehrhardt
IBM Research and Development Germany

11th August 2011
Session 10016
Agenda

• zEnterprise 196 design

• Linux performance comparison z196 and z10

• Compiler case study
z196 Continues the Mainframe Heritage
z196 Water cooled – Under the covers (Model M66/M80) front view

- Internal Batteries (optional)
- Power Supplies
- Support Elements
- I/O drawers
- Flexible Service Processor (FSP) cage controller cards
- Processor Books, Memory, MBA and HCA cards
- InfiniBand I/O Interconnects
- I/O Cards
- 2 x Water Cooling Units
z196 Processor Design Basics

- **CPU (core)**
  - cycle time
  - pipeline, execution order
  - branch prediction
  - hardware vs. millicode

- **Memory subsystem**
  - high speed buffers (caches)
    - on chip, on book
    - private, shared
    - coherency required

- **Buses**
  - number
  - Bandwidth

- **limits**
  - distance + speed of light
  - space

![Generic Hierarchy example](image-url)
z196 vs. z10 hardware comparison

- **z10 EC**
  - **CPU**
    - 4.4 Ghz
  - **Caches**
    - L1 private 64k instr, 128k data
    - L1.5 private 3 MiB
    - L2 shared 48 MiB / book
    - book interconnect: star

- **z196**
  - **CPU**
    - 5.2 Ghz
    - Out-of-Order execution
  - **Caches**
    - L1 private 64k instr, 128k data
    - L2 private 1.5 MiB
    - L3 shared 24 MiB / chip
    - L4 shared 192 MiB / book
    - book interconnect: star
z196 PU core features I

- Super-scalar with six execution units
  - 2 fixed point (integer), 2 load/store, 1 binary floating point, 1 decimal floating point

- Up to five instructions/operations executed per cycle (vs. 2 in z10)
  - Only the *fp units are mutually exclusive
  - Cracking helps to utilize as much units as possible

- Up to three instructions decoded per cycle (vs. 2 in z10)
z196 PU core features II

• 211 complex instructions cracked into multiple internal operations
  • Allows simpler and optimized inner data flow
  • Faster execution than a single complex op by the chance to utilize multiple units
  • 246 of the most complex z/Architecture instructions are implemented via millicode

• Execution can occur out of (program) order
z196 New Instruction Set Architecture

- Re-compiled code gains further performance through 110+ new instructions
  - High-Word Facility (30 new instructions)
    - Independent addressing to high word of 64-bit GPRs
  - Interlocked-Access Facility (12 new instructions)
    - Interlocked (atomic) load, value update and store operation in a single instruction
  - Load/Store-on-Condition Facility (6 new instructions)
    - Load or store; conditionally executed based on condition code
  - Distinct-Operands Facility (22 new instructions)
    - Independent specification of result register (different than either source register)
  - Integer to/from Floating point converts (21 new instructions)
  - Population-Count Facility (1 new instruction)
    - Hardware implementation of bit counting ~5x faster
Significant performance benefits for compute intensive applications through
- reordering, so that stalled instructions don't block following independent ones
- Mitigation of stalls due to result dependencies
- Mitigation of stalls due to storage access

Increase the execution unit utilization
Maintains good performance growth for traditional apps

<table>
<thead>
<tr>
<th>Instrs</th>
<th>In-order core execution</th>
<th>Out-of-order core execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>L1 miss</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>L1 miss</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time

Execution
Storage access

L1 miss

In Orlando 2011
Agenda

• zEnterprise 196 design

• Linux performance comparison z196 and z10

• Compiler case study
z10 vs z196 comparison Environment

• Hardware
  • z196: 2817-718 M49
  • z10 : 2097-726 E26
  • z9  : 2094-718 S18

• Linux distribution with recent kernel
  • SLES11 SP1: 2.6.32.13
  • Linux in LPAR
  • Shared processors
  • Other LPARs deactivated
File server benchmark description

- dbench 3
  - Emulation of Netbench benchmark
  - Generates file system load on the Linux VFS
  - Does the same I/O calls like the smbd server in Samba (without networking calls)
  - Mixed file operations workload for each process: create, write, read, append, delete
  - Measures throughput of transferred data

- Configuration
  - 2 GiB memory, mainly memory operations
  - Scaling processors 1, 2, 4, 8, 16
  - For each processor configuration scaling processes 1, 4, 8, 12, 16, 20, 26, 32, 40
dbench

- dbench as scaling example improves on average by 40%
z10 versus z9

- Improvement z10 versus z9
  - Average improvement with 8 CPUs was 50%
Java benchmark description

- Java server benchmark
  - Evaluates the performance of server side Java
  - Exercises
    - Java Virtual Machine (JVM)
    - Just-In-Time compiler (JIT)
    - Garbage collection
    - Multiple threads
    - Simulates real-world applications including XML processing or floating point operations
  - Can be used to measure performance of CPUs, memory hierarchy and scalability

- Configurations
  - 8 processors, 2 GiB memory, 1 JVM
  - 16 processors, 8 GiB memory, 4 JVMs
  - Java Version 6 SR7
SpecJBB

- Business operation throughput improved by 45%
  - 2 GiB, 8CPU, 1 JVM → +44%
  - 8 GiB, 16 CPU, 4 JVM → + 45%

- Further Service Releases since that:
  - Java Release SR8/9 (2010/11) added 2-4% (z196 toleration)
SpecJBB

- In 2008 the message was +60%
  - With full z10 exploitation it eventually had up to +80%

![Graph showing throughput vs. number of warehouses for SLES10-SP2 results - 1 JVM. The graph compares z9 and z10 with a green line for z10 and a dotted green line for z9. The x-axis represents the number of warehouses ranging from 1 to 20, and the y-axis represents throughput. The graph shows an increasing trend in throughput as the number of warehouses increases.]
CPU-intensive benchmark suite

- Stressing a system's processor, memory subsystem and compiler
- Workloads developed from real user applications
- Exercising integer and floating point in C, C++, and Fortran programs
- Can be used to evaluate compile options
- Can be used to optimize the compiler's code generation for a given target system

- Configuration
  - 1 CPU, 2 GiB memory, Executing one test case at a time
  - N CPUs, executing N same test cases at a time
Compiler

- Linux: Internal driver, kernel 2.6.29, gcc 4.5, glibc 2.9.3
  - Floating Point suite improves by 86%
  - Integer suite improves by 76%

Floating point cases z196 (march=z196) versus z10 (march=z10) improvements [%]

Integer cases z196 (march=z196) versus z10 (march=z10) improvements [%]
z10 versus z9

- Overall improvement about 90%
- Older benchmark suite with other test cases

z10 runtime improvements versus z9 [%]

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Improvement [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testcase 14</td>
<td>100</td>
</tr>
<tr>
<td>Testcase 13</td>
<td></td>
</tr>
<tr>
<td>Testcase 12</td>
<td></td>
</tr>
<tr>
<td>Testcase 11</td>
<td></td>
</tr>
<tr>
<td>Testcase 10</td>
<td></td>
</tr>
<tr>
<td>Testcase 9</td>
<td></td>
</tr>
<tr>
<td>Testcase 8</td>
<td></td>
</tr>
<tr>
<td>Testcase 7</td>
<td></td>
</tr>
<tr>
<td>Testcase 6</td>
<td></td>
</tr>
<tr>
<td>Testcase 5</td>
<td></td>
</tr>
<tr>
<td>Testcase 4</td>
<td></td>
</tr>
<tr>
<td>Testcase 3</td>
<td></td>
</tr>
<tr>
<td>Testcase 2</td>
<td></td>
</tr>
<tr>
<td>Testcase 1</td>
<td></td>
</tr>
</tbody>
</table>
Complex application workload

• Upgrade to IBM zEnterprise 196 provides
  • improvements of throughput
  • reduction of CPU load.
• This improves the throughput driven per CPU by 45%

Oracle Real Application Clusters - z10 versus z196
Agenda

• zEnterprise 196 design

• Linux performance comparison z196 and z10

• Compiler case study
SpecINT2006, 456.hmmer

- Search for patterns in a gene sequence database
- Use of Profile Hidden Markov Models
- Programming Language: C (57 source files)

Function Profile:

<table>
<thead>
<tr>
<th>%Total</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>95.6</td>
<td>P7Viterbi</td>
</tr>
<tr>
<td>1.9</td>
<td>sre_random</td>
</tr>
<tr>
<td>1.8</td>
<td>FChoose</td>
</tr>
</tbody>
</table>
456.hmmer, major part of the hotloop

- Key elements:
  - (Address-)Calculations in **blue**
  - Expressions based on these calculations in **green**
  - Assignments if the expression evaluates to true in **red**

```c
for (k = 1; k <= M; k++) {
    mc[k] = mpp[k-1] + tpmm[k-1];
    if ((sc = ip[k-1] + tpim[k-1]) > mc[k]) mc[k] = sc;
    if ((sc = dpp[k-1] + tpdm[k-1]) > mc[k]) mc[k] = sc;
    if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
    [...]
```
456.hmmer, hotloop, code generated for z10

• Example source code line

```c
if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
```

• GCC generated z10 assembler

```
01: LR GPR3,GPR4
02: LR GPR4,GPR6
03: LG GPR8,176(,GPR15)
04: A GPR4,4(GPR8,GPR1) Problem I Address Generation Interlock on GPR8
05: CR GPR4,GPR3 Note: OOO could help for other non dep. instr.
06: BHRC *+272 Problem II Branch Misprediction
[...]
11: ST GPR4,4(GPR1,GPR2) (non patterned, but frequent change)
12: BRC *-270
```
LOCR, Load on Condition

<table>
<thead>
<tr>
<th>LOCR</th>
<th>R1,R2,M3</th>
<th>RRF-c</th>
</tr>
</thead>
<tbody>
<tr>
<td>'B9F2'</td>
<td>M3</td>
<td>/ / / /</td>
</tr>
</tbody>
</table>

The second operand is placed unchanged at the first operand location if the condition code has one of the values specified by M3; otherwise, the first operand remains unchanged.
456.hmmer, hotloop, code generated for z196

- Example source code line
  
  ```c
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  ```

- GCC generated z196 assembler
  
  ```
  [...]
  A simplified code flow, freed up registers
  01: A GPR10,0(,GPR6) → No Address generation interlock forced
  02: CR GPR1,GPR10
  03: LOCRNHE GPR1,GPR10 No branch, no misprediction
  04: ST GPR1,4(,GPR2)
  ```

- ~70% more throughput for this new code running on z196
- Conditional store would be even better (currently worked on)
  - Free up one more register
  - Prevent the memory from getting written in every loop
    (be aware of coherency needs)
Summary

- z196 performance advantages
  - Higher clock speed
  - More cache
  - OOO processing
  - New instructions
  - More processors
  - Processor scalability

- Some exemplary Performance gains with Linux workloads
  - Up to 45% for Java and complex database
  - Up to 86% for single threaded CPU intense
  - About 40% when scaling processors and/or processes
Questions

- Further information is at
  - Linux on System z – Tuning hints and tips
  - Live Virtual Classes for z/VM and Linux
    http://www.vm.ibm.com/education/lvc/

Christian Ehrhardt
Linux on System z
Performance Evaluation

Research & Development
Schönaicher Strasse 220
71032 Böblingen, Germany

ehrhardt@de.ibm.com
Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.