Performance Characterization of SPEC CPU2006 Integer Benchmarks on x86-64 Architecture

Dong Ye
David Kaeli
Northeastern University

Joydeep Ray
Christophe Harle
AMD Inc.
Outline

• Motivation and background
• Performance characteristics of CPU2006 integer benchmarks on x86-64 (64-bit mode vs. 32-bit mode)
• Program characteristics of selected benchmarks
• CPU2006 vs. CPU2000
Motivation and background

• x86-64 architecture brings 64-bit computing to the PC market
  ➢ Need to evaluate whether PC desktop applications can benefit from 64-bit ISA
• SPEC released its latest CPU suite (CPU2006) last month
  ➢ Want to evaluate how applications have changed when moving from CPU2000 to CPU2006
x86-64: extends x86 to 64 bits

- x86-64 == x64 == (AMD64 + EM64T)
- Fully compatible with existing x86 modes
- Architectural support for 64 bit virtual address space and 52 bit physical address space
- 64-bit mode supports flat addressing
- 64-bit integer operations
- 16 64-bit GPRs, 16 SSE registers
Evaluation environment

- Athlon 64 X2 4400+ Rev E (dual-core, 2.2GHz)
- 2 DIMMs 1GB DDR400 DRAM
- SUSE Linux 9.3 Pro x86-64 edition, run level 3, selected daemons are disabled (kernel 2.6.11.4)
- Benchmark bound to run on a single core
- 64-bit binary run in the 64-bit mode, 32-bit binary run in compatibility mode (referred to as 32-bit mode), both on the same 64-bit OS
- GCC 4.1.1 (-O2 for perlbench, -O3 for all others)
- H/W counters used to collect performance data
How much faster is 64-bit mode?

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>64-bit vs. 32-bit speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>perlbench</td>
<td>C</td>
<td>3.42%</td>
</tr>
<tr>
<td>bzip2</td>
<td>C</td>
<td>15.77%</td>
</tr>
<tr>
<td>gcc</td>
<td>C</td>
<td>-18.09%</td>
</tr>
<tr>
<td>mcf</td>
<td>C</td>
<td>-26.35%</td>
</tr>
<tr>
<td>gobmk</td>
<td>C</td>
<td>4.97%</td>
</tr>
<tr>
<td>hmmer</td>
<td>C</td>
<td>34.34%</td>
</tr>
<tr>
<td>sjeng</td>
<td>C</td>
<td>14.21%</td>
</tr>
<tr>
<td>libquantum</td>
<td>C</td>
<td>35.38%</td>
</tr>
<tr>
<td>h264ref</td>
<td>C</td>
<td>35.35%</td>
</tr>
<tr>
<td>omnetpp</td>
<td>C++</td>
<td>-7.83%</td>
</tr>
<tr>
<td>astar</td>
<td>C++</td>
<td>8.46%</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>C++</td>
<td>-13.65%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td><strong>7.16%</strong></td>
</tr>
</tbody>
</table>
Code size *increases* in 64-bit mode

![Bar chart showing code size increases in 64-bit mode for various benchmarks.](chart.png)
Runtime memory footprint increases in 64-bit mode
Dynamic instruction count decreases in 64-bit mode
IPC comparison

IISWC 2006
Instruction cache (L1) request rate *increases* in 64-bit mode.

![Bar chart showing percentage increase in instruction cache request rate for various benchmarks. The x-axis represents the benchmarks (e.g., perlbench, bzip2, gcc), and the y-axis represents the percentage increase.]
Instruction cache miss rate comparison

![Graph showing instruction cache miss rates for various benchmarks. The x-axis represents different benchmarks: perbench, bzip2, gcc, mcf, gobmk, hmer, sjeng, libquantum, h264ref, omnetpp, astart, xalancbmk. The y-axis represents the number of instruction cache misses per 1K instructions. The bars show the miss rate for 64-bit and 32-bit architectures.](image-url)
Data cache (L1) request rate decreases in 64-bit mode.
Data cache miss rate comparison

- 64-bit
- 32-bit
Observations

- Instruction cache miss rate is very low in both 64-bit and 32-bit modes
- Data cache request rate decreases significantly in 64-bit mode
  - Extra registers help
- Data cache miss rate increases in 64-bit mode
  - The increased size of long and pointer data types has an adverse impact on data cache performance
  - A lower instruction count magnifies this
Memory controller utilization ratio comparison

Percentage

Perbench  Bzip2  Gcc  Mcf  Gobmk  Hmmer  Sjeng  Libquantum  H264ref  Omnetpp  Astar  Xalancbmk

64-bit  32-bit

IISWC 2006 16
### Program characteristics of five selected benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Performance</th>
<th>Observations</th>
<th>Root cause analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcf</td>
<td>64-bit: 26.4% slower</td>
<td>Memory footprint doubles</td>
<td>Larger memory footprint due to extensive uses of longs and pointers</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>64-bit: 13.7% slower</td>
<td>Memory footprint increases by 33.9%</td>
<td>Larger memory footprint due to extensive uses of pointers</td>
</tr>
<tr>
<td>hmmer</td>
<td>64-bit: 34.3% faster</td>
<td>Dynamic instruction count decreases by 8.7%</td>
<td>More registers available in 64-bit mode</td>
</tr>
<tr>
<td>libquantum</td>
<td>64-bit: 35.4% faster</td>
<td>Dynamic instruction count decreases by 54%</td>
<td>Native 64-bit integer arithmetic in 64-bit mode</td>
</tr>
<tr>
<td>h264ref</td>
<td>64-bit: 35.4% faster</td>
<td>Dynamic instruction count decreases by 10%</td>
<td>Faster calling convention (mainly because of more registers) in 64-bit mode</td>
</tr>
</tbody>
</table>
CPU2000int: Speedup in 64-bit mode

Percentage

gzip  vpr  gcc  mcf  crafty  parser  eon  perlbmk  gap  vortex  bzip2  twolf

IISWC 2006
Conclusions

- Programs that use features in 64-bit mode (64-bit integer arithmetic and more registers) stand to benefit more from x86-64
- Programs that are memory intensive or make heavy use of long and pointer need to carefully evaluated when porting to x86-64
All performance numbers referred to in this presentation are ‘estimates’ because they are from a ‘peak-only’ SPEC CPU2006 run, and hence is not fully compliant with SPEC run rules. It is expected, though not proven, that results from a fully compliant run would be very close.
Thank you and questions?