RC25126 (W1103-087) March 14, 2011
Computer Science

IBM Research Report

Load/Store Characteristics for a Set of POWER Systems Benchmarks

José E. Moreira
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Load/Store Characteristics for a Set of POWER Systems Benchmarks

José E. Moreira
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
jmoreira@us.ibm.com

Abstract

We analyze the fraction of loads and stores in a set of POWER Systems benchmarks. The benchmarks include all of the SPECint2006 codes plus a set of commercial codes and a set of Python codes. The analysis is based on instruction traces collected from the execution of each of those benchmarks. We observe that on average the load/store instructions represent 40% of the total number of instructions. We also observe that loads and stores appear at a ratio of approximately 2-to-1.

1 Introduction

We analyze the fraction of loads and stores in a set of POWER Systems benchmarks. The benchmarks include all of the SPECint2006 codes [4] plus a set of commercial codes and a set of Python codes. The commercial codes include SPECjAppServer [5], SPECjbb [6] and TPC-C [7]. For Python, we chose codes from the Unladen Swallow benchmark suite [2].

The analysis is based on instruction traces collected from the execution of each of those benchmarks. We first discuss how those traces were collected and processed to extract the load and store instructions. We then proceed to present and discuss the data from our measurements.

2 Trace collection and processing

The instruction traces for the SPECint2006 and commercial codes were collected by executing the benchmarks on a POWER Systems machine with the POWER5 processor [3]. The traces cover only a subset of the execution of each benchmark, but were selected as to be representative of the entire execution.

The instruction traces for the Python codes were collected by executing the benchmarks on the Mambo simulator [1]. Again, the traces cover only part of the execution of each benchmark.

The traces are stored in an IBM proprietary format, organized in trace records for each instruction. Each instruction trace record contains various information for the execution of that instruction. In particular, it contains the instruction encoding (a 32-bit value) and, for those instructions that access memory, the effective address of the memory location accessed (a 64-bit value).

We processed those traces, examining each trace record. We used the instruction encoding to decide if the instruction was a load or a store. If it was either a load or a store we then retrieved the corresponding effective address. In the process, we generated a condensed trace that contains only three pieces of information for each load or store instruction: a flag indicating if it is a load or a store; an instruction index (counting from the beginning of the original trace); and the effective address. Other (non-load/store) instructions were simply skipped and were not included in the condensed trace.

3 Benchmark characteristics

Table 1 summarizes our findings. For each benchmark, it shows the total number of instructions in the original trace as well as the number of loads and stores in that trace. It also shows the percentage of loads and stores with respect to the total number of instructions.

The benchmarks are grouped by category. The first category are the SPECint2006 benchmarks. The second category are the commercial benchmarks (SPECjAppServer, SPECjbb, and TPC-C). There are four different instances of TPC-C traces, collected from different threads of the same run. Finally, the third category are the Python benchmarks.

4 Discussion

The results show that, on average, 40% of the executed instructions in the benchmarks analyzed are loads and stores. Furthermore, the ratio of loads to stores is close to 2-to-1. Although the fraction of loads among all instructions varies from 20 to 40%, most benchmarks are close to the average value of 26%. Similarly, the fraction of stores varies from 7 to 21%, but most benchmarks are close to the average value of 13%.

Noteworthy SPECint2006 benchmarks include gcc, for which there are more stores than loads, and h264ref, for which more than half the instructions executed are loads or stores. Not surprisingly, all four threads of TPC-C display similar behavior. The three Python benchmarks also display similar behavior. Again, this is not surprising since the execution traces are essentially capturing the behavior of the Python interpreter.

benchmark	number of instructions	number of loads	percentage of loads	number of stores	percentage of stores
astar	99,999,985	31,039,761	31.04%	9,960,309	9.96%
bzip2	99,999,987	27,828,966	27.83%	7,565,973	7.57%
gcc	99,999,988	19,428,625	19.43%	21,485,844	21.49%
gobmk	99,999,988	22,450,732	22.45%	13,404,056	13.40%
h264ref	99,999,985	38,625,929	38.63%	13,787,847	13.79%
hmmer	99,999,987	30,115,001	30.12%	11,001,985	11.00%
libquantum	99,999,982	20,684,842	20.68%	8,500,671	8.50%
mcf	99,999,981	29,591,518	29.59%	9,051,620	9.05%
omnetpp	99,999,987	27,557,778	27.56%	18,172,050	18.17%
perlbench	99,999,986	26,820,270	26.82%	15,819,003	15.82%
sjeng	99,999,989	21,916,143	21.92%	7,531,278	7.53%
xalancbmk	99,999,982	23,717,289	23.72%	10,057,303	10.06%
SPECjAppServer	105,000,001	28,916,151	27.54%	15,317,284	14.59%
SPECjbb2005	105,000,000	21,261,112	20.25%	14,011,727	13.34%
tpcc.16	370,117,613	99,137,413	26.79%	59,874,416	16.18%
tpcc.18	218,775,866	58,942,110	26.94%	35,813,285	16.37%
tpcc.20	289,381,491	76,832,097	26.55%	45,402,659	15.69%
tpcc.22	219,070,689	58,650,561	26.77%	35,592,861	16.25%
python26-linux-gcc	72,822,693	18,319,381	25.16%	10,460,060	14.36%
python-django-linux-64	181,574,950	46,068,183	25.37%	25,238,109	13.90%
2to3-p7-python26-fast	149,327,514	37,605,533	25.18%	20,555,872	13.77%
min			19.43%		7.53%
max			38.63%		21.49%
average ± stddev			26.21 ± 4.17%		13.37 ± 3.57%

Table 1: Summary of results for the benchmarks analyzed.

5 Conclusions

Our analysis of several benchmarks running on POWER Systems show that loads and stores represent between 1/3 to 1/2 of all instructions executed in each of those benchmarks. Loads are more numerous than stores by a 2-to-1 ratio on average, although the SPECint2006 benchmark gcc shows more stores than loads.

References

[1] Patrick Bohrer, James Peterson, Mootaz Elnozahy, Ram Rajamony, Ahmed Gheith, Ron Rockhold, Charles Lefurgy, Hazim Shafi, Tarun Nakra, Rick Simpson, Evan Speight, Kartik Sudeep, Eric Van Hensbergen, and Lixin Zhang. Mambo: a full system simulator for the PowerPC architecture. *SIGMETRICS Perform. Eval. Rev.*, 31:8–12, March 2004.

[2] Google Inc. *Unladen Swallow Benchmarks.* `http://code.google.com/p/unladen-swallow/wiki/Benchmarks`.

[3] R. Kalla, Balaram Sinharoy, and J.M. Tendler. Ibm power5 chip: a dual-core multithreaded processor. *Micro, IEEE*, 24(2):40 – 47, 2004.

[4] Standard Performance Evaluation Corporation. *CINT2006 (Integer Component of SPEC CPU2006)*. `http://www.spec.org/cpu2006/CINT2006`.

[5] Standard Performance Evaluation Corporation. *SPECjAppServer2004*. `http://www.spec.org/jAppServer2004`.

[6] Standard Performance Evaluation Corporation. *SPECjbb2005*. `http://www.spec.org/jbb2005`.

[7] Transaction Processing Performance Council. *TPC-C*. `http://www.tpc.org/tpcc`.