

Pathlengths of SPEC Benchmarks for PA-RISC, MIPS, and SPARC

Larry McMahan and Ruby Lee

Computer Systems Architecture
Hewlett Packard Company
19410 Homestead Road
Cupertino, California 95014

Abstract

The total instruction pathlength and instruction frequency counts are measured for the SPEC89 benchmark programs on the PA-RISC architecture and compared with previously published information for the MIPS and SPARC architectures. The PA-RISC architecture typically requires significantly fewer instructions than the other two architectures for the same benchmark. The differences in counts for instruction sub-classes are compared and used to estimate the actual impact of some PA-RISC architectural features designed for pathlength reduction [2].

1. Introduction

This paper compares the actual instruction pathlengths for the SPEC89 benchmark programs for three different RISC architectures: PA-RISC, MIPS and SPARC. Overall, the number of instructions executed on the PA-RISC architecture is significantly smaller than on the other two architectures.

A previous paper [2] has discussed features of the PA-RISC architecture which contribute to pathlength reduction. This paper measures the actual pathlength reductions in real benchmarks in an attempt to quantify the effectiveness of these architectural features for reducing the dynamic number of instructions needed to execute a given program. This paper concentrates on instruction counts rather than cycle counts to focus on architectural differences in the three architectures, rather than hardware implementation differences. Where the impact is definitely due to compiler differences rather than architectural differences, this is stated in the text.

2. Architectural Differences

PA-RISC, MIPS and SPARC are all RISC architectures with register-based execution, load-store architecture, delayed branching and pipelined single-cycle execution for most instructions. While they have many similarities, a few instruction set differences are listed below. The differences listed are at the time the comparison data was produced in April 1991 [1,5,6,7]. Changes to the architectures since then are noted in brackets and documented in references [8,9,10].

2.1 Load and Store Instructions

Indexed loads: MIPS has only a single (register + displacement) addressing mode. SPARC and PA-RISC both have indexed (register + register) addressing. PA-RISC also has scaled indexing.

Base register modification: Only PA-RISC allows base register modification. The base register may be updated before or after the memory data is accessed.

Load use interlocks: MIPS requires an intervening instruction between the load and the use of a register. SPARC and PA-RISC allow the load of a register to be immediately followed by an instruction which uses it as a source register. The hardware interlocks the pipeline if the data has not yet been loaded. When this data arrives, it is forwarded to the execution-unit for the next instruction which uses it. [Load use interlocks were subsequently added to MIPS.]

Integer to Floating Point: Only MIPS has instructions to move values between integer and floating point registers. SPARC and PA-RISC use load/store sequences.

Double Precision: MIPS had only single precision floating point loads and stores, and requires two instructions to load double precision quantities. SPARC and PA-RISC have double precision loads and stores. [MIPS III added doubleword loads and stores.]

SPARC has single precision floating point register copies only. MIPS and PA-RISC have double precision floating point register copies.

Floating Point Load/Store Displacements: PA-RISC has only 5 bit displacements for floating-point load/store instructions. SPARC displacements are 13 bits and MIPS displacements are 16 bits.

Integer Register Set Differences: All three architectures have 32 32-bit integer registers. SPARC's integer registers are organized in a register window scheme [7]. [MIPS III has 32 64-bit registers.]

Floating Point Register Set Differences: SPARC has 32 32-bit registers or 16 64-bit registers. MIPS has 16 of each. PA-RISC has 56 32-bit registers or 28 64-bit register. [MIPS III has 32 32-bit or 64-bit registers.]

2.2 Branch Instructions

Delay slot nullification: PA-RISC and SPARC allow nullification of the delay slot instruction of a branch.

MIPS requires the delay slot instruction to be executed. [MIPS III allows nullification.]

PA-RISC conditional nullification is on a not-taken backward branch, and on a taken forward branch. SPARC allows nullification only if the branch is not taken.

Conditional Integer Branches: SPARC uses one instruction to set a condition code, and another instruction to branch. MIPS and PA-RISC perform an integer compare and branch in a single instruction. In addition, PA-RISC allows the result of data movement, addition, or a bit test to determine whether a branch is taken. This allows PA-RISC to combine integer computation in the same instruction as the branch.

Conditional Floating Point Branches: All three architectures require 3 instructions for a floating point branch. PA-RISC requires a floating point compare, FTEST, and unconditional branch. Neither MIPS nor SPARC require the FTEST allowing any non floating point compare, but the branch instruction is a floating point conditional branch.

2.3 Integer Computational Instructions

Extract/Deposit: PA-RISC has extract and deposit instructions to manipulate bit fields. These perform the shift right and shift left operations in MIPS and SPARC, but provide more general capabilities, since both ends of the bit-field can be specified. PA-RISC also has an instruction where a bit-field spanning a pair of registers can be shifted into a target register.

Integer Multiply/Divide: Both MIPS and PA-RISC have integer multiply. [SPARC added it later.] MIPS integer multiply instruction operates out of the general registers while PA-RISC's integer multiply operates out of the floating-point registers, since it shares the floating-point multiplier hardware. On the general register side, PA-RISC provides multiply primitives in the form of shift and add instructions, which PA-RISC compilers use to accomplish an integer multiply, often in fewer cycles than on the MIPS integer multiply hardware.

Only MIPS has an integer divide instruction. PA-RISC provides only a divide step instruction. This can save cycles at the cost of more instructions for small divisors.

Predicated Execution: Each PA-RISC computational instruction has a built-in skip operation. That is, the results of a computation can conditionally cause the nullification of the following instruction. [SPARC added a subset of this capability called conditional move in V9.]

2.4 Floating Point Computational Instructions

Multiply and Add, or Multiply and Subtract: PA-RISC has FMPYADD and FMPYSUB instructions, which perform floating multiply and add or multiply and subtract operations concurrently in the same instruction.

FSQRT: MIPS does not have a floating point square root instruction. Both SPARC and PA-RISC do. [MIPS-III added it subsequently.]

3. Source of Data

This section describes the benchmarks used, the techniques used to compile the SPEC89 benchmarks on the PA-RISC architecture, and the data collection tools and methods used to obtain PA-RISC pathlengths and instruction counts.

3.1 Benchmarks Used

The SPEC89 benchmarks were chosen because these were the only benchmarks for which detailed instruction counts for the MIPS and SPARC architectures have been published [1]. Newer SPEC92 benchmarks have been released, but we do not have access to pathlength or instruction count data for architectures other than PA-RISC. Figure 1 is a brief summary of the SPEC89 benchmarks.

Integer Benchmarks		
Name	Lang.	Description
gcc1.35	C	Gnu C Compiler
espresso	C	PLA Optimizer
li	C	Lisp Interpreter (9 Queens)
eqntott	C	Boolean Truth Table Generator

Floating Point Benchmarks		
Name	Lang.	Description
spice2g6	Fort	Analog Circuit Simulator
doduc	Fort	Nuclear Reactor Model
nasa7	Fort	Synthetic Benchmark Kernels
fpppp	Fort	Quantum Chemistry Model
tomcatv	Fort	Vectorized Mesh Generation
matrix300	Fort	Linpack Matrix Program

Figure 1: SPEC89 Benchmark Programs

3.2 Compilers and Hardware Implementations

Both the PA-RISC compilers and hardware implementations were chosen to be ones which were contemporary with those used in [1] to collect the MIPS and SPARC data.

All of the SPEC89 benchmark programs were compiled with the PA-RISC 1.1 C Compiler, version A.08.54, and PA-RISC 1.1 Fortran Compiler, version A.08.05. All programs were compiled with the -O option and linked with the -Wl,-aarchive option. In addition, nasa7 and tomcatv were compiled with the +OP option, matrix300 was compiled with the +OP4 option, and tomcatv was compiled with the -WP,-nv,-ur=4,-ur2=200 option.

The programs were run on the PA-RISC Model 9000 Series 720 workstations. Like MIPS and SPARC, PA-RISC has also made architecture and compiler enhancements since then, but these enhancements were not allowed in the collection of PA-RISC data.

3.3 Data Collection

On all of the SPEC89 programs except gcc1.35 and eqntott pathlengths and instruction counts were collected using the PA-RISC simulator. Information regarding instruction nullification, addressing modes,

register copies, and other detailed information was collected using a filter program which analyzed trace records from the simulator. The simulator was not capable of analyzing gcc1.35 and eqntott because it does not model process forks, which both use. We collected data for these programs using a hardware trace recorder, and were only able to collect total instructions either executed or nullified for these two programs. These two programs are not included in the detailed analysis beyond section 4.

4. Overall SPEC89 Pathlengths

This section compares the overall pathlength for all ten SPEC89 benchmarks on all three architectures. Table 1a shows the total instruction pathlength for each benchmark program, for each architecture. Table 1b shows the total instruction pathlength plus the nullified instructions for PA-RISC and SPARC. For MIPS, it shows total instructions minus the NOP instructions used in place of load-use interlock cycles. While Table 1a is a strict comparison of pathlengths, Table 1b may be a more meaningful comparison of the number of instruction slots used in execution, especially since load-use interlocks have been added to MIPS III. Later references to total pathlengths in this paper refer to Table 1b rather than Table 1a unless noted otherwise.

Benchmark	PA	MIPS	SPARC	M/P	S/P
spice	11570	20114	22872	1.73	1.97
doduc	1051	1613	1303	1.53	1.24
nasa7	5415	9257	6615	1.71	1.22
fpppp	1478	2316	1443	1.57	0.98
tomcatv	796	1813	1626	2.28	2.04
Fp G.M.-m300				1.75	1.43
<i>m300</i>	517	2776	1694	5.37	3.28
<i>Fp G.M.</i>				2.10	1.64
gcc*	1151	1111	1156	0.97	1.00
espresso	1787	2829	2931	1.58	1.64
li	4846	6023	4661	1.24	0.96
eqntott*	1018	1243	1322	1.22	1.30
Int G.M.				1.23	1.19
Tot G.M.-m300				1.49	1.31
<i>Tot G.M.</i>				1.69	1.44

Table 1a: Total Instructios Executed

In both tables, columns 2 through 4 show the total instructions for PA-RISC, MIPS, and SPARC. The PA-RISC totals for gcc and eqntott include nullified instructions because of a tools limitation. Column 5, labelled "M/P", is the ratio of MIPS over PA-RISC instructions (column three divided by column two). Similarly, column 6, labelled "S/P", is the ratio of SPARC over PA-RISC instructions.

Separate geometric means over the six floating-point benchmarks (Fp G.M.), the five floating-point benchmarks excluding matrix300 (Fp G.M.-m300), and the four integer benchmarks (Int G.M.) were taken. The geometric means for all ten benchmarks (Tot G.M.) and all nine benchmarks excluding matrix300 (Tot G.M.-m300) are also shown. Matrix300 gives PA-RISC a huge advantage, and was excluded from the floating-point and total geometric means so as not to overly bias the results in PA-RISC's favor.

Benchmark	PA	MIPS	SPARC	M/P	S/P
spice	11905	17528	23914	1.47	2.01
doduc	1099	1362	1314	1.24	1.20
nasa7	5459	9068	6631	1.66	1.21
fpppp	1486	1925	1447	1.30	0.97
tomcatv	809	1728	1626	2.14	2.04
Fp G.M.-m300				1.53	1.42
<i>m300</i>	517	2338	1695	4.52	3.28
Fp G.M.				1.83	1.63
gcc	1151	1078	1195	0.94	1.04
espresso	1950	2575	3098	1.32	1.59
li	5164	5303	4960	1.03	0.96
eqntott	1018	1231	1444	1.21	1.42
Int G.M.				1.12	1.23
Tot G.M.-m300				1.33	1.33
Tot G.M.				1.50	1.45

Table 1b: Instructions Executed or Nullified

Throughout this paper, all the SPARC data and the detailed instructions counts for MIPS are from [1]. However, the overall totals for MIPS used in Table 1a and 1b are from updated MIPS pathlengths supplied in a posting in the comp.arch newsgroup [3].

Table 1b shows that MIPS and SPARC executed more instructions than PA-RISC for most benchmarks. The only exceptions are gcc for MIPS and fpppp and li for SPARC, where PA-RISC has a pathlength 3-6% longer.

On the integer benchmarks, MIPS executes 12% more instructions and SPARC executes 23% more instructions than PA-RISC. On the FP benchmarks, even after excluding Matrix300 from the floating-point geometric mean, MIPS executes 53% more instructions, and SPARC executes 42% more instructions than PA-RISC. For all the benchmarks, excluding matrix300, both MIPS and SPARC execute 33% more instructions than PA-RISC.

Ignoring other factors, PA-RISC's shorter pathlengths imply that MIPS and SPARC implementations may have to have higher native instruction execution rates (of roughly 33%) to achieve the same effective performance as a PA-RISC implementation. For example, a PA-RISC machine with a native instruction execution rate of 100 MIPS (here, MIPS = Million PA-RISC Instructions Per Second), performs at the same level as a MIPS or SPARC machine with native instruction execution rates of 133 MIPS (here, MIPS = Millions of MIPS or SPARC Instructions Per Second, respectively).

5. PA-RISC Instruction Frequency Counts

The detailed PA-RISC instruction counts for the benchmarks are presented in table A1. The columns are for benchmarks, and the rows for instruction classes and subclasses. All counts are in millions. Five types of counts are distinguished by a tag in the instruction subclass field. These are:

- ' ' (No tag) User-level instruction counts.
- '#' Reference counts included elsewhere.
- '*' Nullified instructions.
- '+' Register load-use interlock cycle counts.

'<' Stall cycle counts not included elsewhere.

Three types of totals are provided for PA-RISC.

Total Actual instructions executed. This is equivalent to the **Total** presented in [1]. This is the data presented in Table 1a.

Total* Actual instructions executed or nullified. This does not appear in [1], but can be obtained for SPARC by adding annulled instructions to **Total**. We believe that it is a closer comparison of equivalent architectural features than **Total**. This is the data presented for PA-RISC in Table 1b.

Total+ Actual instructions executed or nullified plus integer register load use interlocks. This is equivalent to the **Total+** presented in [1] and is included here for reference.

Some of the instruction subclasses are different from those reported in [1]. This is partly due to the architectural differences in PA-RISC, and partly due to differences in capabilities of the tools used. For example, NOPs, nullified instructions, and stalls are counted separately. Nullified instructions are divided into subclasses, to allow separate analysis of the effectiveness of nullification for branching versus predicated execution. Register to register copies is broken out as a separate instruction subclass, to determine how much 'computation' was really data movement. Finally, subclasses of instructions are added for architectural categories not present in either MIPS or SPARC. These were compute and branch instructions, the FTEST instruction, multi-op float instructions, extract and deposit instructions, and shift and add integer computation instructions.

6. Instruction Class Counts

This sections compares the instruction counts in the major instruction classes, and suggest which architectural (or compiler) features contribute towards the differences in these counts. All the data analysis applies to an entire benchmark program, including library routines, since instructions in libraries are not separately counted.

The integer benchmarks gcc1.35 and eqntott are deleted from this study because our tools were not able to provide detailed analysis of these benchmarks. The floating point benchmark matrix300 is shown in the following tables, but it is not included in the either the floating-point geometric means or the total geometric means, because it biases the results too much in favor of PA-RISC. The vastly shorter PA-RISC pathlengths in matrix300 are due mainly to compiler optimizations, rather than due to PA-RISC architectural features.

6.1 Load and Store Instruction Counts

Tables 2a and 2b show the number of load and store instructions executed.

Benchmark	PA	MIPS	SPARC	M/P	S/P
spice	4761	5303	4800	1.11	1.01
doduc	270	450	299	1.67	1.11
nasa7	1732	3411	1879	1.97	1.08
fpppp	570	1124	590	1.97	1.04
tomcatv	242	664	482	2.74	1.99
Fp G.M.-m300				1.82	1.20
<i>m300</i>	<i>234</i>	<i>871</i>	<i>436</i>	<i>3.72</i>	<i>1.86</i>
espresso	572	521	681	0.91	1.19
li	1486	1322	1068	0.89	0.72
Int G.M.				0.90	0.86
Tot G.M.-m300				1.49	1.09

Table 2a: Load Instruction Counts

Benchmark	PA	MIPS	SPARC	M/P	S/P
spice	782	1091	915	1.40	1.17
doduc	117	144	84	1.23	0.72
nasa7	714	1310	738	1.83	1.03
fpppp	238	306	129	1.29	0.54
tomcatv	72	235	189	3.26	2.62
Fp G.M.-m300				1.67	1.04
<i>m300</i>	<i>16</i>	<i>434</i>	<i>217</i>	<i>27.12</i>	<i>13.56</i>
espresso	143	124	144	0.87	1.01
li	902	709	482	0.79	0.53
Int G.M.				0.83	0.73
Tot G.M.-m300				1.37	0.94

Table 2b: Store Instruction Counts

On the floating-point benchmarks, PA-RISC has fewer loads and stores than MIPS or SPARC, but SPARC has significantly fewer stores for doduc and fpppp. The larger number of PA-RISC stores compared to SPARC are caused partly by additional register pressure from the multi-op floating point instructions. Improved register spill optimization reduces the PA-RISC counts with the next generation of compilers. MIPS has larger load and store counts because of its lack of double precision loads and stores. The extremely low PA-RISC numbers on tomcatv and matrix300 result partly from compiler optimizations which avoid register reloads of matrix elements by loop unrolling.

On the integer benchmarks, the higher PA-RISC numbers may be due to higher load/store overhead on procedure calls.

Table 2c shows procedure calls as a percentage of total instructions. Li, the benchmark on which PA-RISC has the highest ratio of loads and stores, also has the highest procedure call density. This is an advantage to the SPARC register-window scheme. If PA-RISC required only one extra load or store per return or call, it would increase the total loads and store on li by 16% and on espresso by 4%. The effects are much smaller on the floating point benchmarks except doduc.

Table 2d shows the number of loads for the floating-point benchmarks if MIPS had double precision loads and stores.

Benchmark	Instr	Calls	Pct
spice	11570	106.0	0.9
doduc	1051	16.8	1.6
nasa7	5415	23.4	0.4
fpppp	1478	1.8	0.1
tomcatv	796	0.0	0.0
<i>m300</i>	<i>517</i>	<i>0.0</i>	<i>0.0</i>
espresso	1787	25.0	1.4
li	4846	326.2	6.7

Table 2.c. Calls and Returns as a Percentage of Total Instructions

Adding double precision loads and stores to MIPS can significantly reduce MIPS floating-point load and store instruction counts.

Benchmark	MIPS loads	M/P	MIPS stores	M/P
spice	4334	0.91	785	1.00
doduc	259	0.96	80	0.68
nasa7	1713	0.99	678	0.95
fpppp	572	1.00	157	0.66
tomcatv	339	1.40	124	1.72
Fp G.M.-m300		1.04		0.94
<i>m300</i>	<i>438</i>	<i>1.87</i>	<i>219</i>	<i>13.69</i>

Table 2d: Hypothetical MIPS Double Loads/Stores

Overall, the primary reasons for the smaller PA-RISC instruction counts for loads and stores are the lack of MIPS double precision loads and stores, and PA-RISC compiler optimizations for matrix operations. The larger PA-RISC floating point register file could also be a factor in its favor on the floating point benchmarks.

6.2 Branch Instruction Counts

Table 3 shows the number of branches executed by the three architectures.

Benchmark	PA	MIPS	SPARC	M/P	S/P
spice	2069	2256	3102	1.09	1.50
doduc	81.4	132	110	1.62	1.35
nasa7	294	315	248	1.07	0.84
fpppp	17.9	22.3	21.4	1.25	1.20
tomcatv	25.3	34.8	38.0	1.38	1.50
Fp G.M.-m300				1.27	1.25
<i>m300</i>	<i>9.7</i>	<i>58.6</i>	<i>57.9</i>	<i>6.04</i>	<i>5.97</i>
espresso	446	448	597	1.00	1.34
li	1216	1230	1108	1.01	0.91
Int G.M.				1.01	1.10
Tot G.M.-m300				1.19	1.21

Table 3: Branch Instructions Executed

MIPS and SPARC execute more branch instructions than PA-RISC in the benchmarks, except for nasa7 and li where SPARC has fewer branch instructions than PA-RISC. In the case of nasa7, PA-RISC executes 116M additional compute and branches (ADDIBF). These may be offset by the reduced number of arithmetic instructions. In the case of li, PA-RISC executes 70M more calls and 70M more returns than SPARC. This is the same number of calls and returns executed by MIPS.

The cause of the reduced branch count for PA-RISC on the other benchmarks varies widely. The significantly

smaller number of branches for PA-RISC in matrix300 and tomcatv are mostly due to loop unrolling by the compiler. In Doduc, PA-RISC has many fewer floating point branches.

In fpppp, PA-RISC has fewer conditional branches than MIPS or SPARC, perhaps due to the predicated execution feature of PA-RISC which allows conditional nullification by integer computation instructions. The number of PA-RISC predicated arithmetic operations (4.2M) is slightly more than the difference in conditional branches between either MIPS or SPARC and PA-RISC (15.0M - 11.6M = 3.4M).

6.3 Integer Computation Instruction Counts

Table 4a shows the integer computation instructions executed on each of the architectures. These include integer arithmetic, logical, shift, extract and deposit, load immediate, and compare instructions.

Benchmark	PA	MIPS	SPARC	M/P	S/P
spice	2863	8614	12860	3.01	4.49
doduc	228	325	425	1.43	1.86
nasa7	1213	1749	1646	1.44	1.36
fpppp	281	49	106	0.17	0.38
tomcatv	105	355	403	3.38	3.84
Fp G.M.-m300				1.29	1.75
<i>m300</i>	<i>23</i>	<i>971</i>	<i>551</i>	<i>42.22</i>	<i>23.96</i>
espresso	608	1395	1472	2.29	2.42
li	1196	1429	1791	1.19	1.50
Int G.M.				1.65	1.91
Tot G.M.-m300				1.38	1.79

Table 4a: Computation Instructions Executed

MIPS and SPARC execute many more integer computation instructions than PA-RISC. The only exception is fpppp, where PA-RISC executes almost three times as many instructions as SPARC and six times as many as MIPS. However, 268M of PA-RISC's 281M integer computation instructions are due to a single opcode, "LDO"! The LDO, or Load Offset instruction, is required when an immediate offset is too large to fit in the static displacement field of a load or store operation. This is an advantage for MIPS and SPARC, which have larger static displacement fields for floating point loads. LDOs also increase the number of PA-RISC integer computation instructions greatly in all the other floating point benchmarks except nasa7 and tomcatv.

Benchmark	LDO pct	M/P	S/P	M'/P'	S'/P'
spice	57.6	3.01	4.49	7.05	10.05
doduc	73.4	1.43	1.86	4.64	4.78
nasa7	28.1	1.44	1.36	1.98	1.75
fpppp	95.4	0.17	0.38	3.50	5.73
tomcatv	27.3	3.38	3.84	3.87	5.04
Fp G.M.-m300				3.88	4.76
<i>m300</i>	<i>79.8</i>	<i>42.22</i>	<i>23.96</i>	<i>221.09</i>	<i>119.78</i>

Table 4b: Effect of LDO's on Integer Computation Instructions

Table 4b shows the effect of giving PA-RISC larger displacements. In this case MIPS and SPARC would execute a factor of three or more integer computational

instructions than PA-RISC.

Column 2 shows the percentage of PA-RISC Integer Computation Instructions that are LDOs. Columns 3 and 4 are the instruction ratios from Table 4a. Columns 5 and 6 show the instruction ratios after PA-RISC has been corrected by removing the extra LDOs. The MIPS and SPARC number have been corrected by having their LUIs and SETHIs removed. Hence, short displacements for floating point load instructions is a major disadvantage for PA-RISC pathlength reduction.

Another anomaly is again matrix300, where MIPS executes 42 times and SPARC executes 24 times more integer computation instructions than PA-RISC. Much of matrix300's integer arithmetic is index calculation, which has been reduced for PA-RISC by loop unrolling. A smaller portion of PA-RISC's advantage in tomcatv is also due to loop unrolling.

The integer computation class of instructions represent the greatest reduction in PA-RISC pathlengths, compared to MIPS and SPARC. Several PA-RISC architectural features, described in [2] contribute to this.

- Indexed Loads and Stores
- Scaled Indexed Loads and Stores
- Address Update Loads and Stores
- Extract and Deposit Instructions
- Shift and Add Instructions
- Combined operation and Branch Instructions

The indexed, scaled, and update addressing modes are the architectural features which are most numerous in the benchmark programs. Table 4c. shows the usage of addressing modes by PA-RISC on the benchmarks. Scaled loads are a proper subset of indexed loads. Address updates are orthogonal and may be used independently of indexing.

Benchmark	Ld +St	Idx	Scale	Addr Updt	Idx Pct	Sc Pct	Updt Pct
spice	5544	3482	3482	251	63	63	5
doduc	387	101	101	47	26	26	12
nasa7	2446	1594	1355	1035	65	55	42
fpppp	808	12	12	6	1	1	1
tomcatv	313	312	312	0	99	99	0
<i>m300</i>	<i>249</i>	<i>221</i>	<i>2</i>	<i>232</i>	<i>89</i>	<i>7</i>	<i>93</i>
espresso	714	252	252	40	35	35	6
li	2389	4	4	251	0	0	10

Table 4c: Indexed, Scaled and Address Updates as a Percentage of Load and Store Instructions

MIPS could require one or two additional integer computation instructions for each indexed load, and both MIPS and SPARC could require an additional integer computation instruction for each scaled or update load.

The programs which show the greatest reduction in PA-RISC integer computation instructions in Table 4a also show a high percentage of either indexed addressing or address update in Table 4c. The integer benchmark, li, which has the least reduction in PA-RISC computation instructions also has the lowest percentage of loads and stores with indexed or update addressing. In fpppp, which was the only benchmark where PA-RISC had more integer computation instructions than MIPS or SPARC, hardly any PA-RISC load and store instructions used either indexed or update addressing.

Table 4d shows the number of extract and deposit instructions, and the number of shift and add instructions, as a percentage of PA-RISC integer computation instructions.

Benchmark	Comp Inst	Ext/Dep	Shift & Add	E/D Pct	S&A Pct	Total Pct
spice	2863	42.1	221.1	1.5	7.7	9.2
doduc	228	12.1	5.8	5.3	2.5	7.8
nasa7	1213	180.9	6.6	14.9	0.5	15.4
fpppp	281	3.2	0.2	1.1	0.0	1.1
tomcatv	105	0.1	0.5	0.1	0.5	0.6
<i>m300</i>	<i>23</i>	<i>0.0</i>	<i>0.1</i>	<i>0.0</i>	<i>0.4</i>	<i>0.4</i>
espresso	608	106.6	85.1	17.5	14.0	31.5
li	1197	127.5	13.8	10.7	1.2	11.9

Table 4d: Extract/Deposits and Shift & Adds as a Percentage of Computation Instructions

These instructions are used less frequently than the addressing features, but still constitute a significant percentage of integer computation instructions. They are more numerous in the integer benchmarks, and in those floating point benchmarks with a lot of integer index calculation and index comparison.

Each extract instruction can replace two instructions when it is used for a shift and mask operation. Each deposit instruction can replace five instructions when it is used to insert a field in a word.

When a PA-RISC shift and add instruction combines a shift operation and an add operation in a single instruction, it reduces the pathlength by one. However, when PA-RISC simulates an integer multiply with a sequence of shift and add instructions, the number of integer computation instructions increase relative to a single MIPS integer multiply instruction. Here, there is an anomaly where more instructions can result in fewer execution cycles. For example, 5 PA-RISC single-cycle shift and add instructions may be used to simulate an integer multiplication in a program, that takes 12 cycles to execute on the MIPS R2000/R3000 with a single integer multiply instruction.

Benchmark	Int Inst	Compare Branch	Compute Branch	Compare Pct	Compute Pct
spice	2863	1781.0	58.5	62.2	2.0
doduc	228	43.3	0.1	19.0	0.0
nasa7	1213	135.3	124.4	11.2	10.3
fpppp	281	11.6	0.2	4.1	0.1
tomcatv	105	12.3	0.0	11.7	0.0
<i>m300</i>	<i>23</i>	<i>1.2</i>	<i>8.5</i>	<i>37.0</i>	
espresso	608	359.3	55.9	59.1	9.2
li	1197	755.6	30.1	63.1	4.9

Table 4e: Combined Compute and Branches as a Percentage of Computation Instructions

Table 4e shows the number of compare and branch and compute and branch instructions as a percentage of integer computation instructions.

While either compare and branch or compute and branch could save an instruction for PA-RISC versus SPARC, only compute and branch can save an instruction compared to MIPS.

The integer computation class of instructions accounts for the greatest portion of the pathlength reduction for PA-RISC, as compared to MIPS and SPARC. Of the architectural features which could contribute to this reduction, scaled indexed loads and stores are the most heavily used, followed in order by address updates on loads and stores, extract and deposit instructions, and finally compute and branch instructions.

6.4 Floating Point Instruction Counts

For floating-point instructions, the primary architectural difference between PA-RISC and the other two architectures is its floating point multiply and add (FMPYADD) and multiply and subtract (FMPYSUB) instructions. Both PA-RISC and SPARC also have a floating point square root instruction, but these are seldom used in the SPEC89 benchmarks.

Table 5a shows the number of floating point computation instructions executed for each architecture. It does not include floating-point load and store instructions.

Benchmark	PA	MIPS	SPARC	M/P	S/P
spice	1027	1091	967	1.06	0.94
doduc	353	430	338	1.22	0.96
nasa7	1460	2289	2075	1.57	1.42
fpppp	370	612	592	1.65	1.60
tomcatv	353	507	501	1.44	1.42
Fp G.M.-m300				1.36	1.23
<i>m300</i>	235	432	432	1.84	1.84

Table 5a: Floating Point Instruction Counts

PA-RISC executes significantly fewer floating-point instructions than MIPS or SPARC for nasa7, fpppp, tomcatv and of course, matrix300.

Benchmark	MIPS mpy+ add	SPARC mpy+ add	PA mple	MIPS minus PA mple	SPARC minus PA mple	PA mpy+ add+ mple
spice	742	681	57	685	624	688
doduc	247	228	49	198	179	182
nasa7	2082	2049	708	1374	1341	1399
fpppp	583	571	222	361	349	353
tomcatv	442	442	102	340	340	294
<i>m300</i>	432	432	197	235	235	235

Table 5b. Pathlength Reduction Effects of FMPYADD and FMPYSUB Instructions

Looking at Table 5b, the PA-RISC reduction in floating point instruction counts is approximately equal to the number of FMPYADD or FMPYSUB instructions executed. Columns 2 and 3 show the floating point multiplies plus adds plus subtracts executed by MIPS and SPARC. Column 3 shows the number of PA-RISC FMPYADD and FMPYSUB instructions. Each instruction represents a 2:1 reduction. If both MIPS and SPARC were able to execute the same number of FMPYADD or FMPYSUB instructions, their number of floating multiplies plus floating adds plus floating subtracts (columns 4 and 5) would be about the same as PA-RISC (column 6).

However, PA-RISC executes slightly more instructions when a large number of floating point branches are

required. This can be seen in spice and doduc. This is because PA-RISC requires two instructions (an FCOMP and an FTEST) for each floating-point conditional branch test. The rationale behind this was to minimize the amount of information that may have to be sent from a floating-point unit that may be in a separate chip, to the integer unit where the branch logic resides. The floating-point compare (FCOMP) performs a floating-point operation and sets a condition bit, which is then tested by a FTEST, which conditionally nullifies the following unconditional branch instruction. A hypothetical fbranch instruction would have reduced the three instructions (FCOMP, FTEST, Branch) to the pair of instructions (FCOMP, fbranch). However, this would have complicated the conditional branch logic since it would need to stall for the floating point condition to be generated. The FTEST instruction is used to absorb any pipeline stalls due to having to wait for the FCOMP instruction to complete. Both SPARC and MIPS solve this problem by requiring at least one intervening instruction, which is not an FCOMP, between the FCOMP and the floating point branch.

Overall, FMPYADD/FMPYSUB is a primary contributor to pathlength reduction for PA-RISC in floating point programs with many inner-product computations or other expressions which contain roughly equal number of multiplications and additions or subtractions.

6.5 NOP Usage and Nullification

Compared to the previous instruction classes, NOPs account for the smallest instruction counts in all three architectures. A very large reduction in this class would cause only a small change in overall pathlength.

Benchmark	PA	MIPS	SPARC	M/P	S/P
spice	386	569	1141	1.47	2.96
doduc	49	87	46	1.77	0.94
nasa7	45	21	21	0.47	0.47
fpppp	8.6	8.4	7.0	0.98	0.81
tomcatv	13.0	6.5	13.2	0.50	1.02
Fp G.M.-m300				0.90	1.02
<i>m300</i>	0.2	0.02	0.7	0.10	3.50
espresso	192	341	176	1.78	0.92
li	355	681	364	1.92	1.03
Int G.M.				1.85	0.97
Tot G.M.-m300				1.11	1.01

Table 6: NOPS and Nullification

Table 6 shows the total number of NOPs and nullified instructions for each architecture. MIPS load-use NOPs have been subtracted out, since we are not comparing register load-use stalls for PA-RISC and SPARC.

PA-RISC does show some larger NOP and nullification counts on nasa7, fpppp, and tomcatv, but these are programs with the smallest total number of NOPs and nullifications for all three architectures. PA-RISC shows smaller counts on spice and li, the two benchmarks with the largest number of NOPs and nullifications for all three architectures.

Overall, PA-RISC shows a slight reduction in the number of NOPs and nullifications, particularly in programs where they occur more frequently.

7. Summary

Benchmark	Int pct	Fp pct	Br pct	Ld pct	St pct	Nop pct
spice	24.0	8.6	17.4	40.0	6.6	2.8
doduc	20.8	32.1	7.4	24.6	10.7	4.4
nasa7	22.2	26.8	5.4	31.8	13.0	0.8
fpppp	18.9	24.9	1.2	38.4	16.0	0.6
tomcatv	13.0	43.6	3.1	29.8	8.9	1.6
Fp A.M.-m300	19.8	27.2	6.9	32.9	11.1	2.1
m300	4.4	45.5	1.8	45.3	3.0	0.0
espresso	31.3	-	22.9	29.3	7.4	9.1
li	23.3	-	23.5	28.8	17.5	6.9
Int A.M.	27.3	-	23.2	29.0	12.5	8.0

Table 7a: PA-RISC Instruction Classes as Percent of Total Instruction Counts

Table 7a shows the percentage of PA-RISC instructions executed in each class. The arithmetic means of the percentage of instructions in each class is calculated to get a "typical" distribution of instructions by class.

For the floating point benchmarks, we see that loads are typically the most common, followed by floating point computation, integer computation, stores, branches, and NOPs, in that order. For the integer benchmarks, we see that loads are typically the most common, followed by integer computation, branches, stores, and NOPs, in that order.

Instr Class	M/P F.P	S/P F.P	M/P Int	S/P Int	M/P Tot	S/P Tot
	G.M.	G.M.	G.M.	G.M.	G.M.	G.M.
Int	1.29	1.75	1.65	1.91	1.38	1.79
Float	1.36	1.23				
Branch	1.27	1.25	1.01	1.10	1.19	1.21
Load	1.82	1.20	0.90	0.86	1.49	1.09
NOP	0.90	1.02	1.85	0.97	1.11	1.01
Store	1.67	1.04	0.83	0.73	1.37	0.94
Total	1.53	1.42	1.12	1.23	1.33	1.33

Table 7b: Geometric Means of Instruction Count Ratios by Instruction Class

Table 7b shows the geometric means of the MIPS/PA-RISC and SPARC/PA-RISC instruction count ratios in each instruction class. This is summarized from Tables 1b, 2a, 2b, 3, 4a, 5a, and 6. The greatest reduction in instruction counts came in the class of integer computation instructions, followed by floating-point instructions, branch instructions, load instructions, NOPs and nullified instructions, and store instructions, in roughly that order.

MIPS executed 38% more integer computation instructions than PA-RISC, while SPARC executed 79% more integer computation instructions, based on the total geometric means for this class of instructions. Looking at the detailed instruction counts, of the architectural features which could cause the reduction in integer computation instructions, scaled indexed loads was the most heavily used, followed by address updates, extract and deposit instructions, compute and branch, and shift and add instructions. However, PA-RISC's small static displacements for floating-point load and store instructions also increased the number of LDO instructions, which were counted as integer computation

instructions. The high percentage of PA-RISC load and store instructions which use addressing modes beyond the vanilla "base plus displacement" addressing mode in MIPS indicates the usefulness of indexed and update addressing modes.

The reduction in PA-RISC floating point instructions correlated well with the number of FMPYADD and FMPYSUB instructions used. However, this was partially offset by the number of FTEST instructions required by PA-RISC in programs which performed many floating-point branch tests.

PA-RISC performed about 20% fewer branches than MIPS or SPARC overall. However, we could not definitively isolate the cause, although predicated execution remains a possible contributor.

PA-RISC also executed slightly fewer NOPs and nullified instructions. However, we could not find a definitive reason.

Finally, after compensating for MIPS lack of double precision floating point loads, PA-RISC executed slightly fewer loads, but slightly more stores than MIPS and SPARC.

8. Conclusions

We measured the total pathlength and instruction counts for the SPEC89 benchmarks on PA-RISC using machines and compilers contemporary to those used in generating the data published in [1]. Overall, both MIPS and SPARC needed approximately the same or more instructions than PA-RISC on each SPEC89 benchmark. The total geometric mean of the ratio of pathlengths, even after excluding matrix300, showed that both MIPS and SPARC executed 33% more instructions than PA-RISC. Since these programs are large and different in nature, this may be considered a significant empirical measure of the effectiveness of the PA-RISC pathlength reduction features in both its instruction-set architecture and its compilers.

A number of architectural changes as well as compiler changes have occurred in all three architectures, since the time this data was collected. Nevertheless, this paper represents a detailed baseline comparison of the pathlengths and instruction class counts for PA-RISC, MIPS, and SPARC, at a given point in time, for a significant set of programs. It also provides some quantitative measure of the effectiveness of specific PA-RISC architectural features for pathlength reduction. In addition, it can be used to estimate the effects of architectural and compiler changes.

9. References

- [1] Cmelik, R.F., et. al., "An Analysis of MIPS and SPARC Instruction Set Utilization on the SPEC Benchmarks," *ASPLOS-IV Proceedings*, April 1991, pages 290-302.
- [2] Lee, R., et. al., "Pathlength Reduction Features in the PA-RISC Architecture," *Proceedings of IEEE COMPCON Spring 1992*, Feb 1992, pages 129-135.
- [3] Mashey, J., comp.arch posting, "Re: SPARC implementation or architecture", 20 Apr 1991.

- [4] Lee, R., "Precision Architecture," *IEEE Computer*. Vol. 22, No. 1, January 1989.
- [5] "PA-RISC 1.1 Architecture and Instruction Set Reference Manual, 1st Edition," Part Number 09740-90039, Hewlett Packard, November 1990.
- [6] Kane, Gerry, *MIPS R2000 RISC Architecture*, Prentice Hall, Englewood Cliffs, NJ, 1987.
- [7] "The SPARC Architecture Manual, Version 7," Part Number 800-1399-08, Sun Microsystems, Inc., Mountain View, California, 1987.
- [8] "PA-RISC 1.1 Architecture and Instruction Set Reference Manual, 2nd Edition," Part Number 09740-90039, Hewlett Packard, November 1992.
- [9] "MIPS R4000 Microprocessor User's Manual," Part Number M8-00040, MIPS Computer Systems, Inc., Sunnyvale, California, 1991.
- [10] "The SPARC Architecture Manual, Version 8," Part Number 800-1399-12, Sun Microsystems, Inc., Mountain View California, 1990.

Appendix A: Instruction Class Counts

This appendix describes the legends used in Table A1: PA-RISC Instruction Counts by Opcode Class.

Loads and stores: Any memory reference instruction

- int: Integer GR Load or Store
- fp dp: Double Precision Floating Point Load or Store
- fp sp: Single Precision Floating Point Load or Store
- indexed: Any indexed load
- scaled: Any scaled indexed load (subset of indexed)
- update: Any load or store which updates the address base register

Branches: Any control transfer instruction

- compare: Integer compare and branch (includes BB)
- compute: Integer compute and branch (ADDxBx, MOVxBx)
- uncond: Unconditional branch not in the following classes
- fp cond: Unconditional branch following an FTEST instruction
- call: Unconditional branch used for procedure call
- ret: Unconditional branch used for procedure return

Integer: Any Integer computation instruction

- copy: Register to register copy
- arith: Integer arithmetic
- logical: Boolean arithmetic
- ext/dep: Extract or deposit instruction
- shift: Shift instruction
- load imm: Load immediate instruction (LDO, LDIL, ADDIL)
- compare: Compare and clear instruction (COMCLR, COMICLR)
- shiftadd: Shift and add instruction (subset of arith)
- predicated: Integer computation instruction which is capable of conditionally nullifying the following instruction.

Floating Pt.: Any floating point computation instruction.

- int mpy: (32x32=64bit) Integer multiply which is performed in the floating point unit.

- dp add/sub: Double precision add or subtract.
- dp mpy: Double precision multiply
- dp div: Double precision division
- dp cmp: Double precision compare
- dp multiple: FMPYADD or FMPYSUB which performs multiplication independently with addition or subtraction.
- dp other: Other double precision (square root and absolute value)
- sp: Any single precision instruction
- convert: Any instruction to convert among fixed, double, or single precision.
- ftest: FTEST (see description in text)
- dp cpy: Double precision register copy
- sp cpy: Single precision register copy

No-ops: Any NOP. Almost all PA-RISC NOPs occur after branches.

Other: PA-RISC system control instructions. (less than 0.5%)

Nullified: PA-RISC instructions which were nullified by the previous instruction. The nullifying instruction is listed in the subclasses.

- cond br: A conditional compare and branch which nullified the subsequent instruction
- comp br: A conditional compute and branch which nullified the subsequent instruction
- uncond br: An unconditional branch which nullified the subsequent instruction.
- call/ret: A call or return which nullified the subsequent instruction.
- ftest: An FTEST which nullified the subsequent instruction.
- fp br: An unconditional branch following an FTEST which nullified the subsequent instruction.
- predicated: An integer computation which nullified the subsequent instruction.

Stalls: Register load-use interlock cycles.

- int load: A Stall cycle caused by an integer general register load.
- fp load: A Stall cycle caused by a floating point register load.