CPU2006 Working Set Size

Darryl Gove
Systems Group, Sun Microsystems Inc.
Contact: darryl.gove@sun.com

Introduction

SPEC CPU2000 had a target memory footprint of 200 MB for the benchmarks [1], to enable the suite to run on machines with 256 MB of memory. Six years have elapsed since the release of that suite, and in that time memory sizes have increased significantly, so the memory requirements for the recently released CPU2006 reflect this. CPU2006 has been targeted to have a benchmark memory footprint of about 900MB, allowing the suite to run on machines with 1GB of memory.

There are multiple metrics of memory usage. The most common are the *Virtual Size* (*VSZ*) (called "vsz" by the Unix '`ps`' utility) and the *Resident Set Size* (*RSS*) (called "rss" by '`ps`'). The VSZ and the RSS for the benchmarks in the suites are summarised in this paper, and examined in greater depth in [2].

VSZ is the amount of address space that the operating system has reserved for the application. This memory is a set of address ranges available to hold data or instructions; the addresses do not actually have to hold useful data, or even be resident in physical memory (for example they could be paged out to disk).

The *RSS* is a measure of how much physical memory is actually being used by the application; often this will be the same as the *VSZ* of the application, but it may be less. When a computer runs low on physical memory, parts of applications can be paged out to disk, reducing the *RSS* for that application, but keeping the same *VSZ*.

The VSZ of an application does not necessarily reflect the minimum memory required to run the application. As an example, suppose that an application starts, and has an initial *VSZ* of 100MB, and all of this data is present in physical memory. The application would also have an *RSS* of 100MB because the data is resident in memory. The next thing the application does is to call the Unix `mmap` routine to allocate space in memory to read a file from disk. This file has a size of 1GB, so the Unix `mmap` call will reserve 1GB of virtual memory, but will not read the contents of the file into this memory until the contents are actually used. The *VSZ* of the application would reflect the fact that the application is using 1.1GB of address space. However, the *RSS* would remain at 100MB, since none of the file has yet been mapped into memory. If 100MB of the file is read into memory, and then the file is closed, the *RSS* will increase to 200MB, but the *VSZ* will remain at 1.1GB.

As can be seen from the example, the *VSZ* of an application is always greater than the *RSS* of the application, and having a large *VSZ* does not necessarily mean that all of that physical memory is needed to run the application.

Memory usage is also affected by libraries and optimizations. For example, an application might call an optimized library for memory management (e.g. the Unix calls `malloc` and `free`) to improve the speed of `malloc` and `free` at the expense of using a larger amount of memory. In this situation the application code is not changed, but the new library will cause a larger memory footprint. Similarly there are some compiler optimizations which use memory to hold temporary copies of data. These optimizations trade some increased memory usage for an improvement in runtime.

The metrics of *VSZ* and *RSS* are appropriate measures of the amount of memory that an application occupies. However they give only limited insight into how much memory an application actually uses. For example it is relatively easy to allocate arrays to hold the largest data set that the application can handle, but when the application is run on a smaller data set, these arrays take up memory, but contain no useful data.

The *Working Set Size* (*WSS*) is an estimate of how much memory is actually being actively used by an application. The difference between *VSZ*, *RSS*, and *WSS* can be conveyed by further consideration of the earlier example.

Recall that after reading 100MB of data from a file of 1GB in size, the 100MB application has a *VSZ* of 1.1GB, and an *RSS* of 200MB. Suppose the application uses the 100MB of data read in from a file to perform some calculation. This calculation iterates over the 100MB of data for the entire runtime of the application to eventually produce a single value as output. If no other data is touched, it can be argued that the application has a *WSS* of 100MB.

The idea of *WSS* is related to measurements of cache performance (such as miss rates). However WSS has the big advantage that it is independent of any cache implementation in hardware.

Related work

The idea of *Working Set Size* appears in multiple contexts. Denning [3] was responsible for early work in this area, evaluating WSS in the context of reducing the number of page faults as memory was paged to and from disk.

Sair and Charney [4], evaluate the idea of *WSS* in relationship to TLB misses when running CPU2000; they determine that a 4MB pagesize substantially reduces the number of TLB misses.

In another context, Cantin and Hill [5] look at the idea of *WSS* at the level of the L1 caches and evaluate the decline in the number of cache misses as the cache size increases. Similarly Hallnor and Reinhardt [6] investigate the impact of data compression on the WSS of the CPU2000 benchmarks.

Methodology and Working Set Size

To gather the data presented in this paper, the SPEC CPU benchmarks were compiled on an UltraSPARC-III based system using the Sun Studio 11 compiler with the `-fast` optimization flag (`-fast` produces 32-bit binaries by default). The values for *RSS* and *VSZ* were the end of run values reported by `spot`[7] for each workload.

As has been indicated by the section on related work, the idea of a *Working Set Size* can be applied at cacheline size or at the TLB page size levels, depending on what the objective is. To determine a good trade-off for cache size, the *WSS* needs to be tracked at the level of cachelines. To determine how big the TLB needs to be, the *WSS* needs to be tracked at the level of page sizes. This paper focuses on the WSS at the level of the 64-byte block, since many processors have adopted this size at some level in their cache hierarchy.

The estimates for *WSS* were obtained using a SHADE [8] based tool written for this purpose. Shade allows execution tracing with a variety of user-written analysis tools. In this case the tool tracked the address of every load and store that an application performed. The memory addresses were tracked at the level of 64-byte blocks. An array was used to record the particular blocks touched. After an interval of 1 billion memory operations, the array was traversed to determine how many blocks had been touched during the interval. The *WSS* for the interval was calculated by multiplying the number of blocks touched in the interval by the size of each block (64 bytes). This paper reports the average *WSS* over the entire run of the workload.

The sampling interval will have an impact on the *WSS* reported. The longer the interval the more likely it is that the workload will have touched all of its useful data - hence a long interval should lead to a larger estimate for the WSS. The interval of 1 billion memory operations was selected because a billion memory operations will take a time that can be measured in units that are on the *order of* seconds. For example, if one in every four instructions is a load or a store, then 1 billion memory operations is roughly 4 billion total instructions. If one instruction is executed every cycle on a 1GHz processor then it will take four seconds of real time to complete the 1 billion memory operations. An alternative view of the interval of 1 billion memory operations is that if each memory operation was fetching 4-bytes from a different memory location, then the application could have touched 4GB of memory in that interval. This value is much greater than the 900MB footprint that SPEC was targeting for CPU2006.

It would be useful to get some indication of 64-byte blocks that are repeatedly touched versus blocks that are rarely touched. Hence, further results are also captured from the trace. A measure of *Core Working Set Size (CWSS)* is also calculated, which is a measure of the number of 64-byte blocks that were touched in the current sample that were also touched in the previous sample. This is indicative of the number of blocks that are regularly used. These blocks that are regularly accessed would benefit from being held in cache, whereas blocks that are used and then discarded are not likely to be worth holding in cache. This paper reports the average *CWSS* over the entire run of the workload.

In order to evaluate the progression from CPU2000 to CPU2006, the *VSZ*, *RSS, WSS,* and *CWSS* were captured for all workloads in CPU2000 and CPU2006. A workload in this context means one single run of a benchmark under a unique set of parameters. For example, the benchmark 164.gzip in CPU2000 has five workloads, and it is the total time for completing all five workloads which is reported as the result for performance runs of the benchmark suite.

Results of VSZ and RSS analysis

The charts shown in the following results have a common layout. The x-axis shows the memory size in MB, the y-axis shows the percentage of workloads that would need more than that memory size. For example a memory size of 2GB is sufficient to contain any of the workloads, hence the percentage needing more than that size would be 0.

The *VSZ* and *RSS* are a measure of the amount of physical memory which is necessary to load the workload and its data. If the benchmark is run on a system will insufficient physical memory then the system will end up paging memory to disk during the run of the benchmark. The *VSZ* and *RSS* determine the minimum physical memory that is needed to run the suite without paging data to disk.

The *VSZ* and *RSS* for the Integer part of the CPU2000 and CPU2006 suites are shown in Figure 1. The *VSZ* and the *RSS* for the Integer workloads in the suite are similar. For CPU2000 all the Integer workloads fit into a memory size of 256MB. For CPU2006, all the Integer workloads will fit into 1GB of memory. For both CPU2000 and CPU2006, a memory size of 32MB would fit about 20% of the Integer workloads. However, very few workloads in the CPU2006 Integer suite will fit into less than 16MB.

The *VSZ* and *RSS* data for the Floating Point workloads are presented in Figure 2. In this case there is a significant difference between the *VSZ* and *RSS* of the workloads because some of the Floating Point workloads reserve more memory than they actually need. A result of this is that several workloads reserve more than 1GB of virtual memory, however there are no workloads with an RSS of greater than 1GB. About 60% of the workloads in CPU2000 need more than 32MB of memory, in contrast about 90% of the workloads in CPU2006 need more than 32MB.

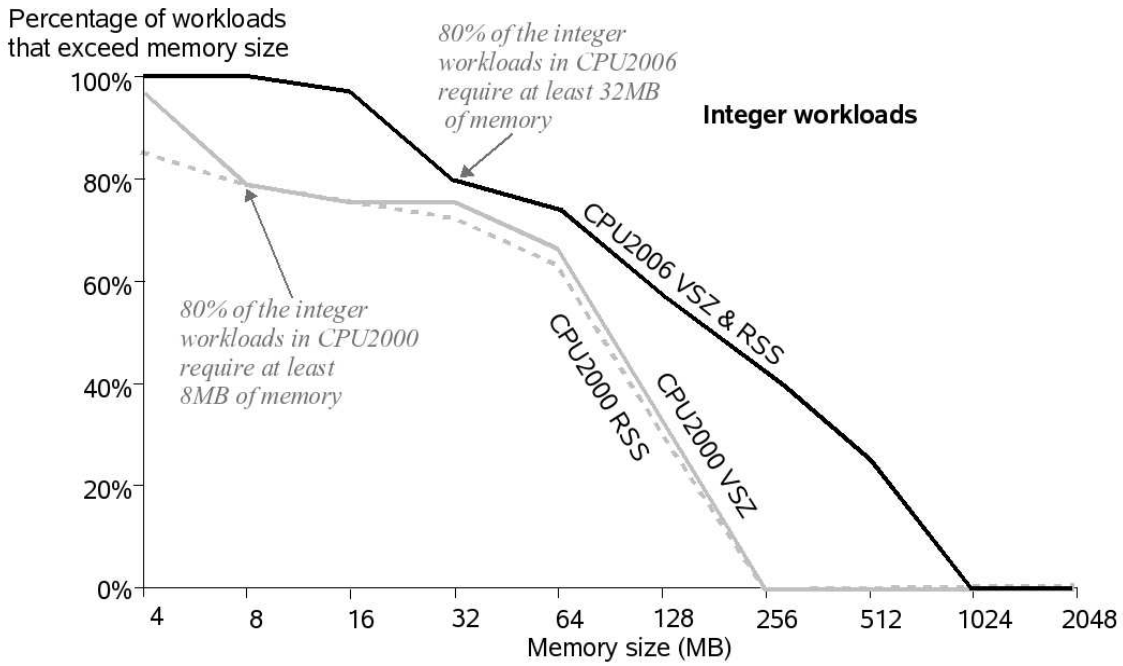

Figure 1: VSZ and RSS for the Integer workloads

From Figures 1 and 2 it is readily apparent that the memory footprint has substantially increased going from CPU2000 to CPU2006. From the chart it is possible to determine that about 60% of the Integer and 40% Floating point workloads in CPU2006 would fit into the memory requirements for the older CPU2000 suite.

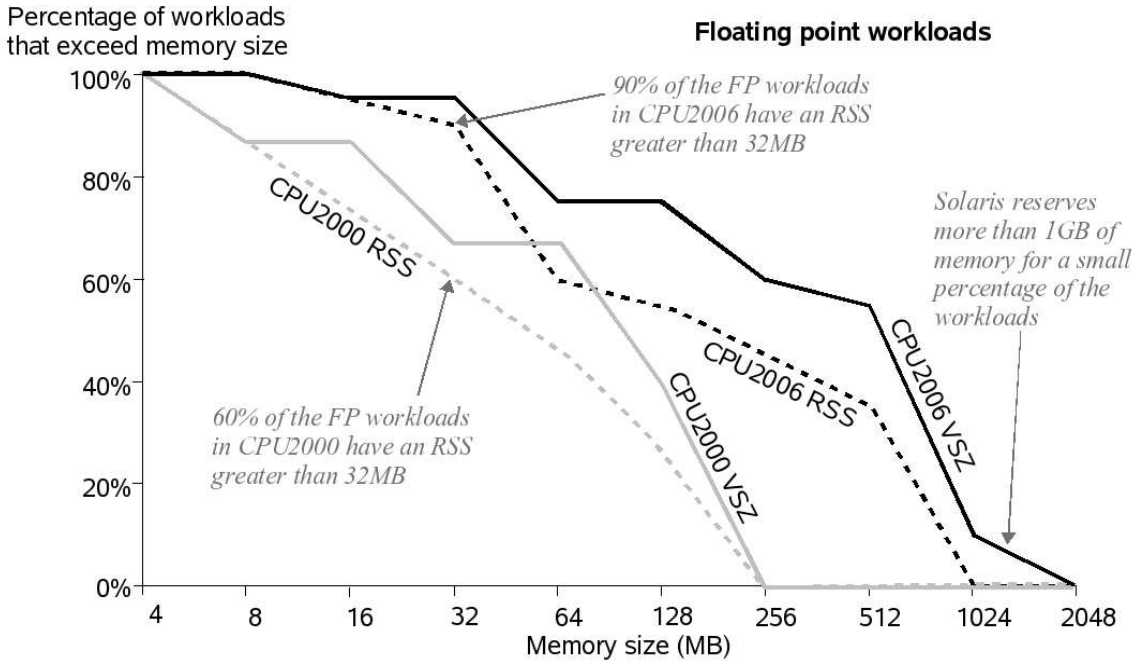

Figure 2: VSZ and RSS for the Floating Point Workloads

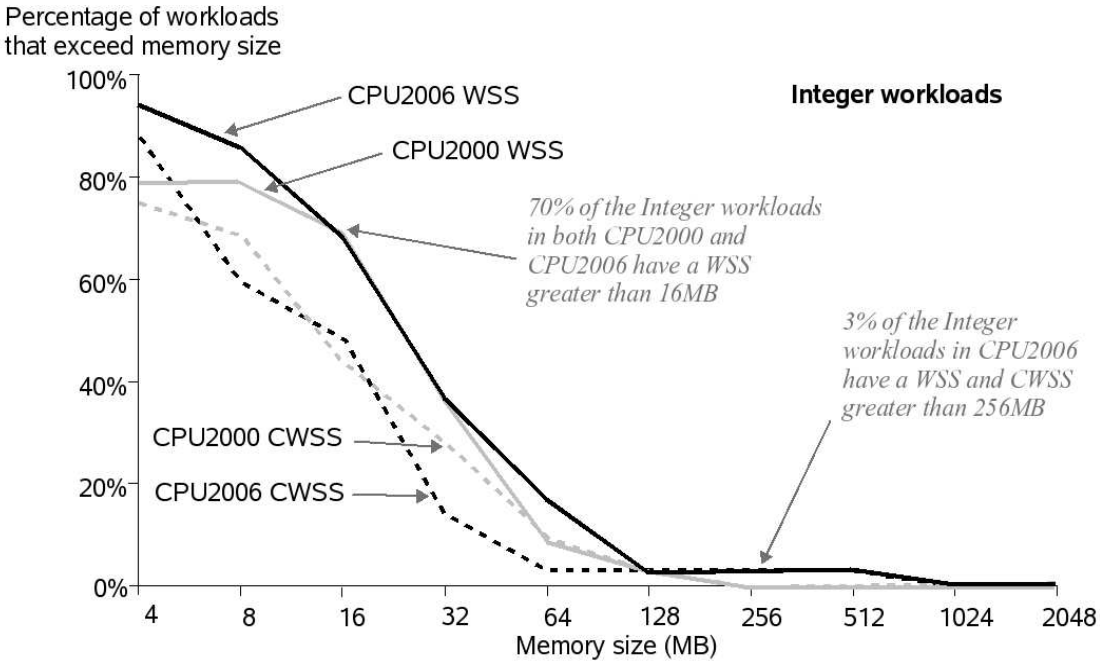

Figure 3: WSS for the Integer workloads

Results of WSS analysis

This section contains the results of *WSS* analysis of the workloads.

Figure 3 shows the *WSS and CWSS* for the Integer benchmarks in CPU2000 and CPU2006. The proximity of the *WSS* results for CPU2000 and CPU2006 may seem surprising.

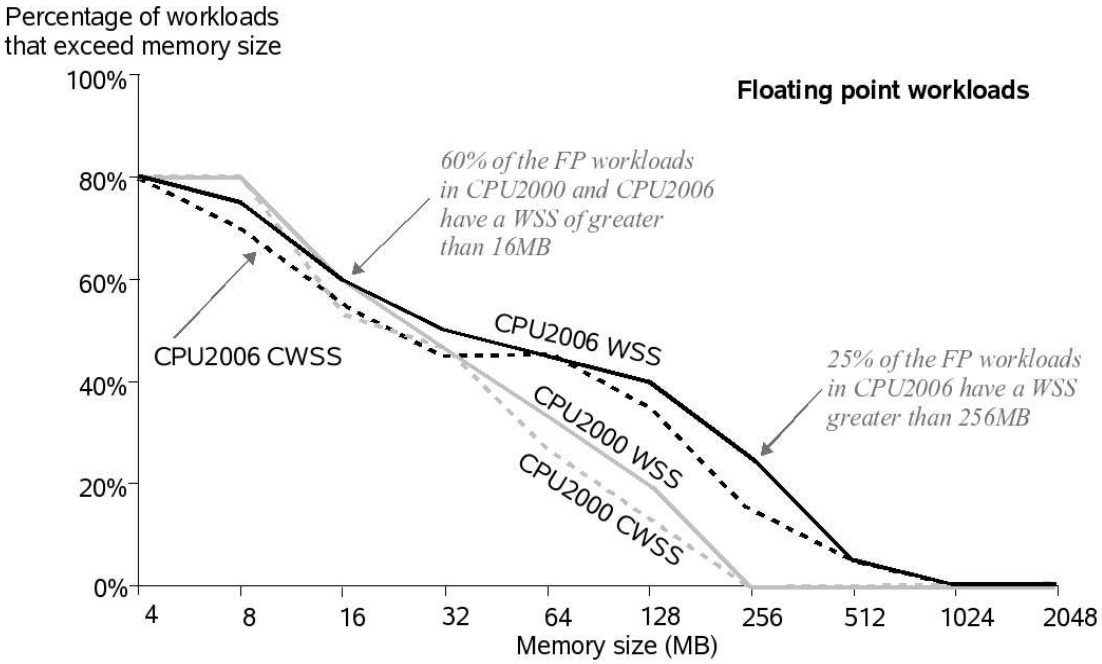

Figure 4: WSS for the Floating Point workloads

All the Integer workloads in CPU2000 and most of the Integer workloads in CPU2006 have a *WSS* of less than 256MB. One workload in CPU2006, 429.mcf, has a *WSS* much greater than 256MB. The biggest difference is at the low end of the scale where about 20% of the workloads in CPU2000 have a *WSS* of less than 4MB, compared to about 5% of the workloads in CPU2006.

It appears that the overall WSS is very similar across the two suites, however CPU2006 has a lower proportion of workloads with smaller *WSS* and a small proportion of workloads with a *WSS* greater than 256MB - the *WSS* has increased at both the low and high ends.

Figure 4 shows the *WSS* and *CWSS* for the Floating Point workloads in CPU2000 and CPU2006. Comparing the two suites about half of the workloads have a similar *WSS*. However, for Floating Point there has been a substantial increase in *WSS* of the other half of the workloads. About 25% of the Floating Point workloads in CPU2006 have a *WSS* greater than 256MB, in comparison the largest *WSS* for CPU2000 was 171.swim with 201MB .

Interestingly, 20% of the Floating Point workloads in both CPU2000 and CPU2006 have a *WSS* of less than 4MB. In CPU2000 this group contains 179.art and 200.sixtrack. For CPU2006 the Floating Point benchmarks with a *WSS* below 4MB are 416.gamess and 453.povray. This is a different result from the Integer workloads where CPU2006 shows a smaller proportion of Integer workloads with a *WSS* of less than 4MB.

The raw WSS and CWSS results are reported in tables 1 to 4, together with the standard deviation for the two metrics. Cells where there is insufficient data are denoted by N/A.

Sample results of WSS evolution over time

Since the procedure for obtaining *WSS* reports results every 1 billion memory operations, it is useful to examine these results over time in order to determine behaviour of the process over their entire run. This section shows the results from workloads that show different behaviour over time.

Figure 5 shows that for the benchmark 453.povray (which has only one workload) the *WSS* is stable at around 0.4MB. The standard deviation, for 453.povray, calculated for both the *WSS* and the *CWSS* is 0.1 MB.

Figure 6 shows the *WSS* for the benchmark 447.dealII (which has only one workload). This has an average *WSS* of 41.7MB and *CWSS* of 25MB, but it is apparent that these numbers come from a pattern of an increasing *WSS* coupled with lots of reuse of a smaller part of that *WSS*. This is reflected in the standard deviation for the *WSS* and *CWSS* for 447.dealII is which is significantly greater than the mean value.

Figure 7 shows the WSS over time for 447.namd, this is reasonably representative of the behaviour of WSS for most of the benchmarks. There is some considerable variation in WSS between samples. A small working set size indicates considerable reuse of particular memory blocks.

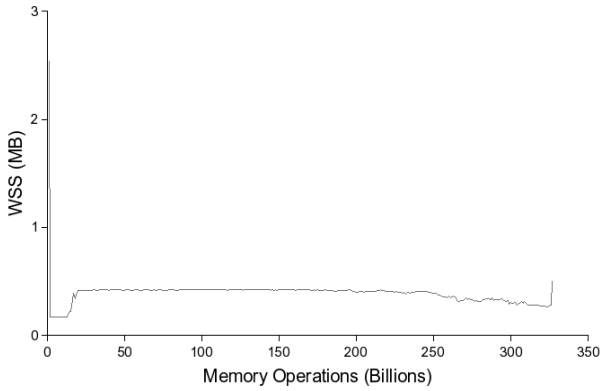

Figure 5: WSS over time of 453.povray

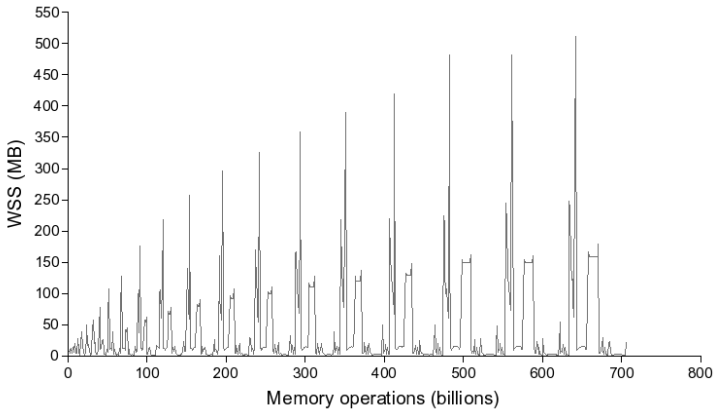

Figure 6: WSS over time for 447.dealII

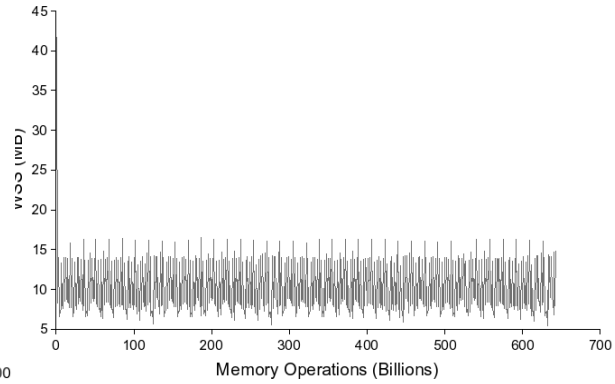

Figure 7: WSS over time for 447.namd

Concluding remarks

This paper presents data showing that the target of requiring a larger memory footprint for CPU2006 over the older CPU2000 suite has been met for *VSZ* and *RSS*.

The paper uses a measure of *Working Set Size* (*WSS*) to examine whether this increase in memory requirement is also reflected in an increase in memory used. It appears that the memory used has increased, but the increase is due to:

- Fewer Integer workloads with small memory footprint. Only 5% of the workloads in CPU2006 have a WSS of less than 4MB, compared to 20% of those in CPU2000.
- An increase in the number of Floating Point workloads with a large memory footprint. 25% of the Floating Point workloads in CPU2006 have a *WSS* of greater than 256MB.

These results should not be unexpected. It is relatively easy to increase memory footprint for Floating Point workloads because these often have a direct relationship between the size of the input data and the amount of memory that needs to be traversed to calculate the answer.

The most surprising result is that there is a higher proportion of Floating Point workloads that have a WSS of less than 4MB than Integer workloads. In particular the benchmark 416.gamess has a WSS which is about 1000x smaller than its VSZ. Similarly 453.povray has the smallest RSS in the suite at only 9MB. Neither of these applications fit the traditional view of Floating point workloads.

For Integer workloads the relationship is less straightforward, in fact it can be argued that the relative stability of the *WSS* for Integer workloads between the two suites may indicate a feature of Integer workloads in general rather than some kind of bias or selection effect in the Integer part of the suite.

Note: These results were not available during the preparation of the suite.

Acknowledgments

Many thanks to those who contributed to this analysis. In particular I appreciate the assistance of John Henning and Partha Tirumalai who suggested a number of refinements to the draft of this paper.

References

[1] John L. Henning, "SPEC CPU2000 Memory Footprint", http://www.spec.org/cpu2000/analysis/memory/

[2] John L.Henning, "SPEC CPU2006 Memory Footprint", Computer Architecture News, this issue, also available at www.spec.org/cpu2006

[3] Denning, P. J. The working set model for program behaviour. Communications of the ACM, Vol 11, Issue 5. May 1968.

[4] Suleyman Sair, Mark Charney, "Memory behavior of the SPEC2000 benchmark suite", IBM Research Report RC 21852, 2000

[5] Jason F Cantin, Mark D Hill, "Cache performance for selected SPEC CPU2000 benchmarks", ACM SIGARCH Computer Architecture News, v29, issue4, September 2001.

[6] Hallnor E. G., Reinhardt S. K. A unified compressed memory hierarchy. High-Performance Computer Architecture, 2005. HPCA-11, 2005.

[7] Simple Performance Optimisation Tool. http://cooltools.sunsource.net/spot/

[8] R. F. Cmelik and D. Keppel. Shade: a fast instruction-set simulator for execution profiling. In Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pages 128–137, May 1994.

Tables

Benchmark	VSZ (MB)	RSS (MB)	WSS (MB)	Std	CWSS (MB)	Std
400.perlbench	198	193	6.4	5.7	5.3	3.0
	333	330	21.1	30.4	1.7	8.3
	594	591	51.3	25.7	25.1	13.8
401.bzip2	877	871	24.4	45.0	5.3	2.5
	111	105	14.0	9.8	6.1	2.1
	111	108	10.6	8.2	5.9	2.0
	877	873	21.5	40.4	4.9	2.3
	877	873	16.4	39.7	4.5	3.5
	631	628	24.4	45.0	5.3	3.5
403.gcc	247	244	65.9	47.4	30.4	19.3
	210	207	69.2	29.7	59.8	26.1
	443	439	57.1	71.4	23.2	28.8
	321	316	50.9	65.6	21.2	28.9
	439	436	73.0	106.0	29.3	42.6
	595	592	67.4	95.3	26.3	40.3
	849	846	70.7	119.3	25.0	45.7
	960	955	37.6	16.9	33.6	15.1
	91	89	37.6	16.9	33.6	15.1
429.mcf	865	865	680.8	241.9	616.8	307.6
445.gobmk	30	29	16.5	1.9	15.7	2.5
	30	29	15.8	3.2	14.2	4.0
	30	29	16.2	2.4	15.7	2.9
	30	29	16.8	1.6	16.2	2.1
	30	29	14.6	4.5	12.1	5.4
456.hmmer	13	13	8.2	4.6	6.1	2.5
	62	62	2.0	0.2	1.9	0.1
458.sjeng	185	185	57.7	17.4	29.1	9.2
462.libquantum	108	107	32.7	4.9	32.3	3.5
464.h264ref	34	33	8.4	0.7	5.0	0.5
	26	25	5.5	0.6	4.1	0.7
	71	70	6.2	2.9	2.1	0.5
471.omnetpp	125	124	24.1	5.3	21.0	3.0
473.astar	321	314	26.0	18.7	22.0	11.4
	137	136	3.5	3.8	3.1	2.3
483.xalancbmk	351	345	27.8	17.8	20.1	11.3

Table 1: - Memory footprint for the CPU2006 Integer workloads

Benchmark	VSZ (MB)	RSS (MB)	WSS (MB)	Std	CWSS (MB)	Std
410.bwaves	917	900	474.3	131.4	429.5	33.3
416.gamess	684	36	0.6	0.4	0.5	0.3
	684	37	0.4	0.3	0.3	0.1
	684	39	1.3	1.0	1.1	0.7
433.milc	693	691	230.8	58.9	212.5	28.5
434.zeusmp	1167	533	270.1	55.9	182.9	35.5
435.gromacs	41	26	8.6	0.2	8.6	0.0
436.cactusADM	1044	761	307.1	7.9	206.5	6.3
437.leslie3d	147	133	75.2	1.8	75.2	0.0
444.namd	55	54	10.2	3.4	5.5	1.1
447.dealII	579	577	14.7	68.3	25.0	47.3
450.soplex	141	126	27.2	7.9	24.3	5.1
	641	442	201.5	30.6	196.6	33.6
453.povray	10	9	0.4	0.1	0.4	0.1
454.calculix	239	225	23.9	23.9	8.2	10.0
459.GemsFDTD	868	854	800.0	15.0	800.0	15.0
465.tonto	63	46	6.2	7.4	4.8	5.8
470.lbm	427	427	402.0	13.6	400.3	29.0
481.wrf	737	715	163.5	47.9	120.6	34.2
482.sphinx3	50	50	10.6	1.3	9.8	1.1

Table 2: - Memory footprint for CPU2006 Floating Point workloads

Benchmark	VSZ (MB)	RSS (MB)	WSS (MB)	Std	CWSS (MB)	Std
164.gzip	186	186	56.7	25.5	38.0	29.8
	186	186	62.9	13.5	55.4	17.1
	186	186	96.7	20.7	78.3	18.0
	186	186	105.0	18.5	91.8	15.7
	186	186	40.5	32.6	19.0	24.7
175.vpr	5	4	1.2	0.3	1.2	0.0
	42	41	29.8	2.7	27.2	1.9
176.gcc	151	150	62.5	47.5	28.6	25.6
	98	97	22.4	14.0	13.5	6.6
	52	51	35.6	14.3	24.2	N/A
	73	72	56.8	10.5	47.2	N/A
	94	93	21.4	16.7	11.0	8.2
181.mcf	99	98	74.0	26.9	41.3	25.2
186.crafty	4	3	1.4	0.1	1.3	0.0
197.parser	33	24	13.6	3.3	11.6	2.6
252.eon	4	3	0.1	0.2	0.1	0.0
	4	3	0.1	0.2	0.2	0.0
	4	3	0.1	0.2	0.1	0.0
253.perlbmk	72	71	12.4	14.4	1.0	1.2
	11	10	8.2	N/A	N/A	N/A
	3	2	0.1	0.1	0.1	0.0
	120	119	23.3	17.4	10.2	8.2
	64	63	19.5	12.0	10.4	9.1
	66	65	20.6	12.8	11.6	9.5
	96	95	21.0	13.5	10.4	8.5
254.gap	200	199	174.3	27.9	168.2	34.3
255.vortex	87	86	44.5	11.4	33.6	9.4
	68	67	31.3	10.4	19.3	4.1
	96	95	46.5	14.3	32.6	8.8
256.bzip2	191	190	26.4	20.9	10.0	8.2
	191	190	25.6	18.8	7.0	3.4
	191	190	25.2	18.8	7.5	4.0
300.twolf	5	4	1.2	0.3	1.2	0.0

Table 3: - Memory footprint for the CPU2000 Integer workloads

Benchmark	VSZ (MB)	RSS (MB)	WSS (MB)	Std	CWSS (MB)	Std
168.wupwise	196	182	162.3	17.7	161.9	17.8
171.swim	215	201	68.3	19.9	57.4	17.3
172.mgrid	72	59	55.0	0.2	54.6	0.0
173.applu	210	67	63.3	0.1	63.3	0.0
177.mesa	24	11	8.0	0.5	8.0	0.5
178.galgel	172	64	15.7	6.1	12.1	5.7
179.art	5	4	3.4	0.3	2.2	0.0
	5	4	2.4	0.3	2.2	0.0
183.equake	30	28	20.6	1.2	20.4	0.4
187.facerec	70	35	16.3	1.7	15.9	0.8
188.ammp	16	15	13.2	0.2	13.2	0.3
189.lucas	161	148	142.2	0.1	142.2	0.0
191.fma3d	124	109	98.7	13.0	98.0	16.0
200.sixtrack	77	30	1.5	2.0	1.2	0.0
301.apsi	211	198	136.9	15.2	104.5	20.5

Table 4: - Memory footprint for CPU2000 Floating Point workloads