CPU 2006 Working Set Size

Darryl Gove, Sun Microsystems (now Oracle)

ACM SIGARCH Computer Architecture News
Vol. 35, No. 1, March 2007

Metrics of Memory Usage – Vsz and RSS

1. Vsz
2. Virtual Size
3. The amount of address space that the operating system has reserved for the application.
4. RSS
5. Resident set size
6. A measure of how much physical memory is actually being used by the application.
7. RSS may be often same as vsz or it may be less.

Metrics of Memory Usage – Vsz and RSS

Vsz

Virtual Size

The amount of address space that the operating system has reserved for the application.

RSS

Resident set size

A measure of how much physical memory is actually being used by the application.

RSS may be often same as vsz or it may be less.

Metrics of Memory Usage – Vsz and RSS

Application has an initial footprint of 100M

VSZ?

100M

RSS?

100M

WSS?

Application calls Unix mmap to allocate space for reading a 1GB file

VSZ?

1.1GB

Metrics of Memory Usage – Vsz and RSS

Application reads first one tenth of the file and is working on it

VSZ?

1.1GB

RSS?

200M

WSS?

Depends on what all are accessed – Max = 200M

Application iterates over the $1/10^{th}$ of the file multiple times but accesses nothing else

VSZ = 1.1GB; RSS = 200M; WSS = 100M

Differences in malloc

Normal malloc and free

Solaris optimized library's malloc and free

Optimized version takes larger amount of memory

But optimized for performance

Normal version reserves less memory

Working Set

WSS or working set size

WSS is an estimate of the size of memory in use during application execution. It is usually estimated over a small interval.

First Reference to WSS

Denning 1968

How is WSS estimated in Gove 2007?

At the level of 64-byte blocks

An array is used to record particular blocks touched. After 1 billion mem operations, array traversed

The Working Set Model for Program Behavior

Peter J. Denning

Massachusetts Institute of Technology, Cambridge, Massachusetts

Probably the most basic reason behind the absence of a general treatment of resource allocation in modern computer systems is an adequate model for program behavior. In this paper a new model, the "working set model," is developed. The working set of pages associated with a process, defined to be the collection of its most recently used pages, provides knowledge vital to the dynamic management of paged memories. "Process" and "working set" are shown to be manifestations of the same ongoing computational activity; then "processor demand" and "memory demand" are defined; and resource allocation is formulated as the problem of balancing demands against available equipment.

KEY WORDS AND PHRASES: general operating system concepts, multiprocessing, multiprogramming, operating systems, program behavior, program models, resource allocation, scheduling, storage allocation

develop a new model, the *working set model*, which embodies certain important behavioral properties of computations operating in multiprogrammed environs, enabling us to decide which information is in use by a running program and which is not. We do not intend that the proposed model be considered "final"; rather, we hope to stimulate a new kind of thinking that may be of considerable help in solving many operating system design problems.

The working set is intended to model the behavior of programs in the general purpose computer system, or computer utility. For this reason we assume that the operating system must determine on its own the behavior of programs it runs; it cannot count on outside help. Two commonly proposed sources of externally supplied allo-

tion about other modules may be unavailable at compilation time. Because of data dependence there may be no

Presented at an ACM Symposium on Operating System Principles, Gatlinburg, Tenn., October 1-4, 1967; revised November, 1967. Work reported herein was supported in part by Project MAC, an M.I.T. research project sponsored by the Advanced Projects Research Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01).

[1] There have been attempts to do this. Ramamoorthy [2], for example, has put forth a proposal for automatic segmentation of programs during compilation.

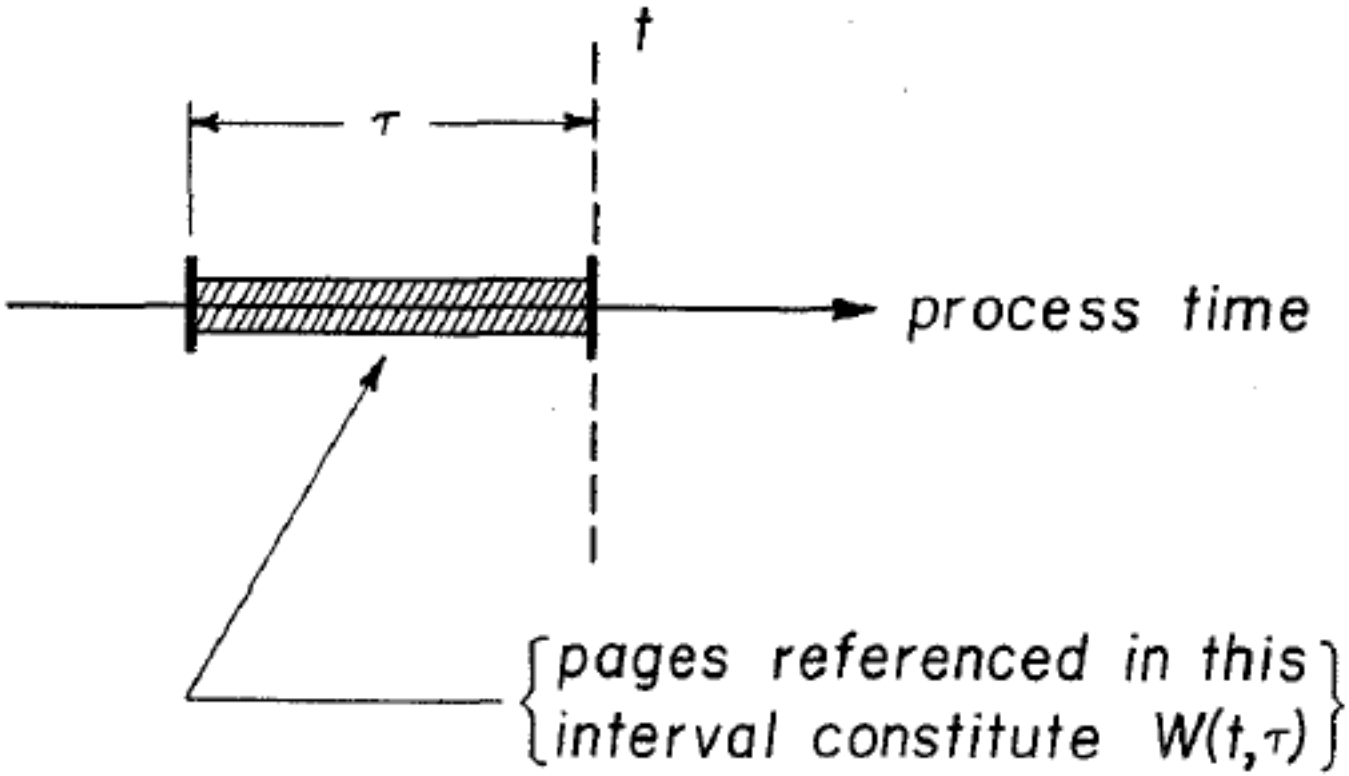

FIG. 2. Definition of $W(t, \tau)$

Denning's Definition of Working Set

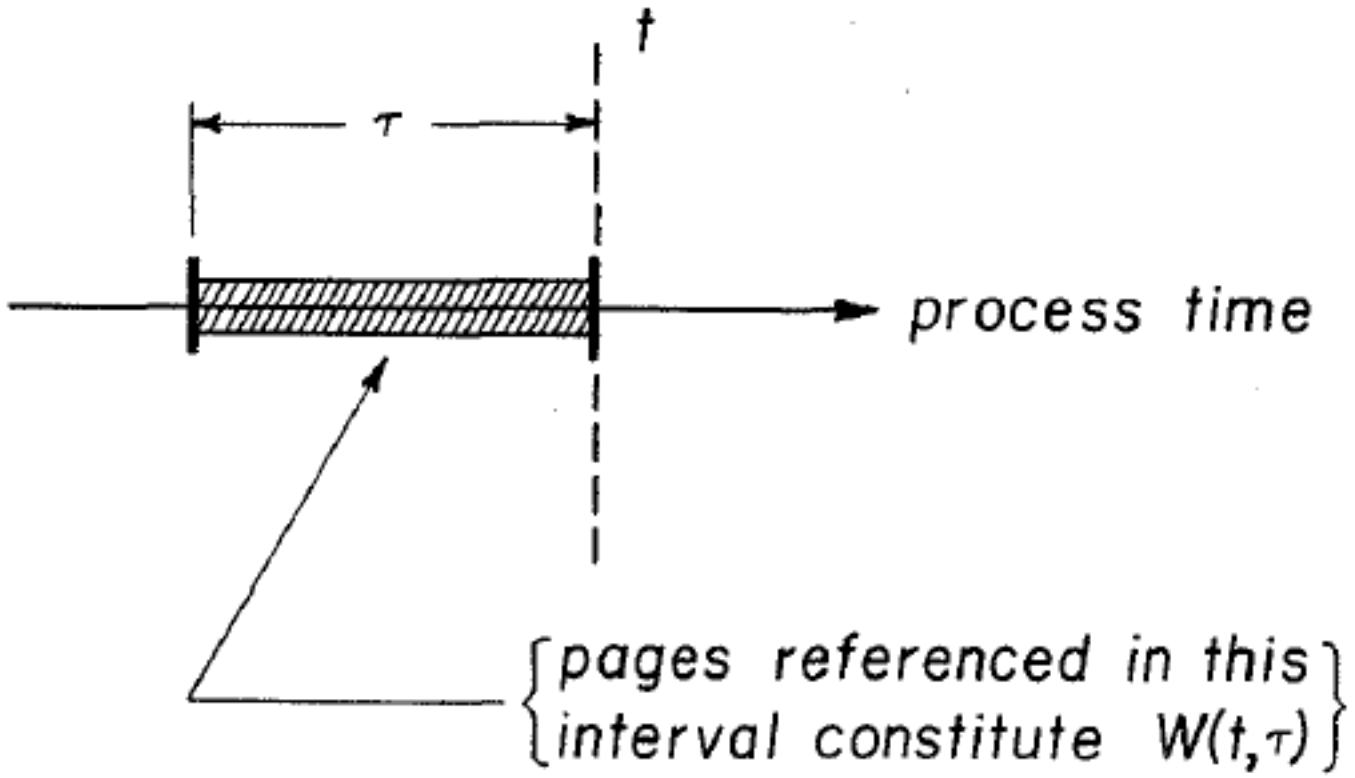

FIG. 2. Definition of $W(t, \tau)$

Working Set

Different ways to estimate WSS

Granularity

Interval

Block size/page size

What is the significance of 1 billion mem operations used by Gove?

Core Working Set Size (CWSS)

Blocks touched this interval and last interval

Why should one measure CWSS in addition to WSS?

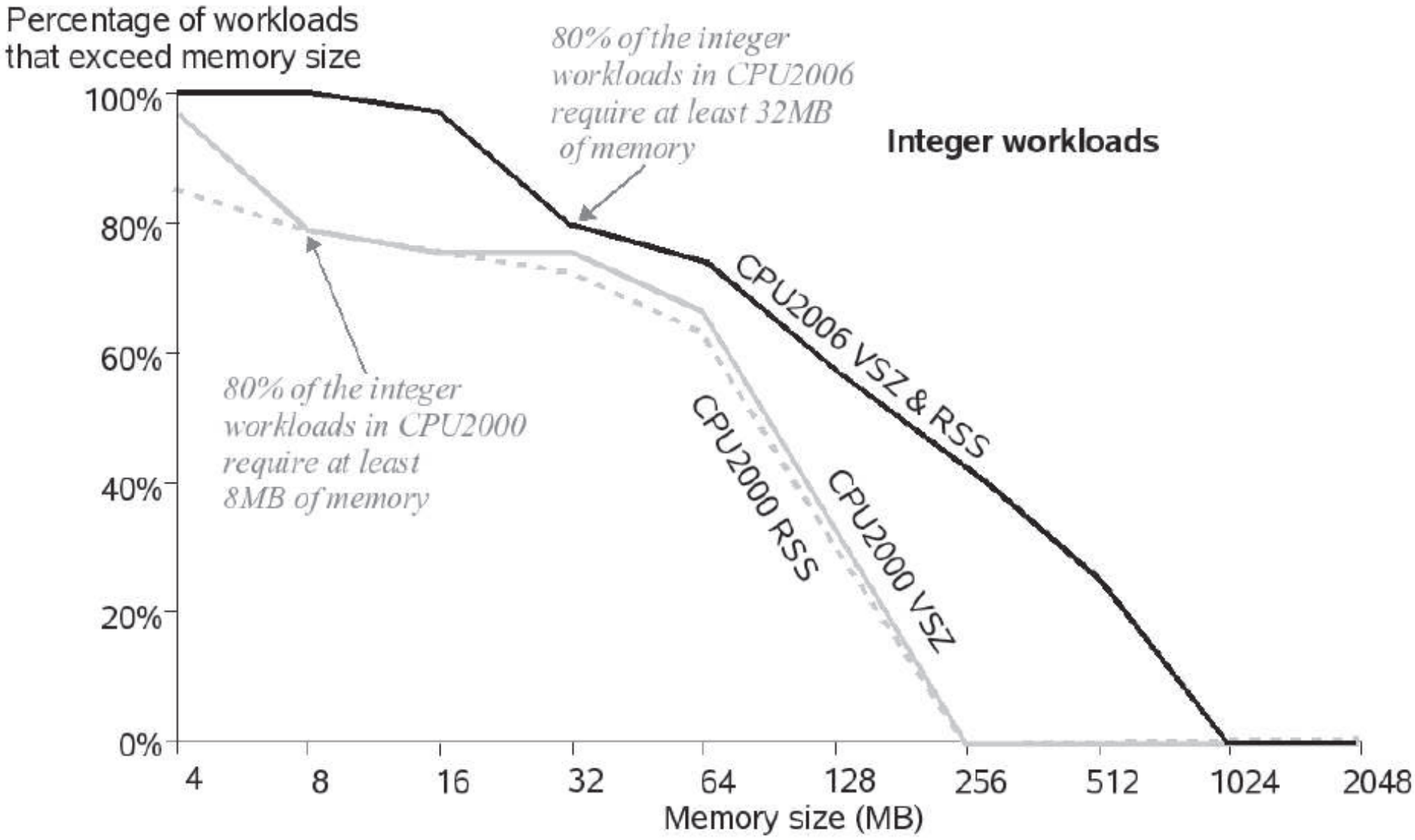

Figure 1: VSZ and RSS for the Integer workloads

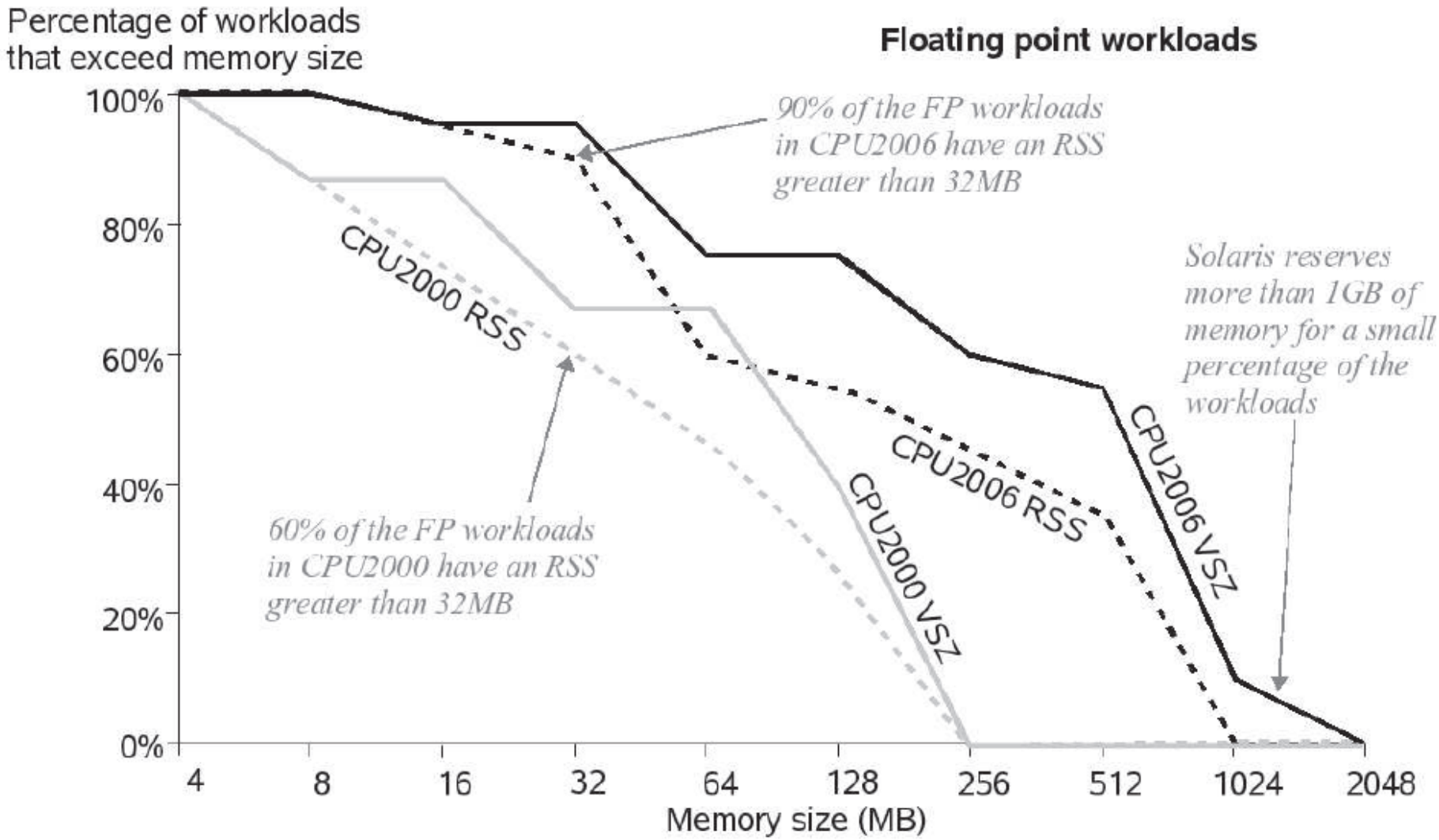

Figure 2: VSZ and RSS for the Floating Point Workloads

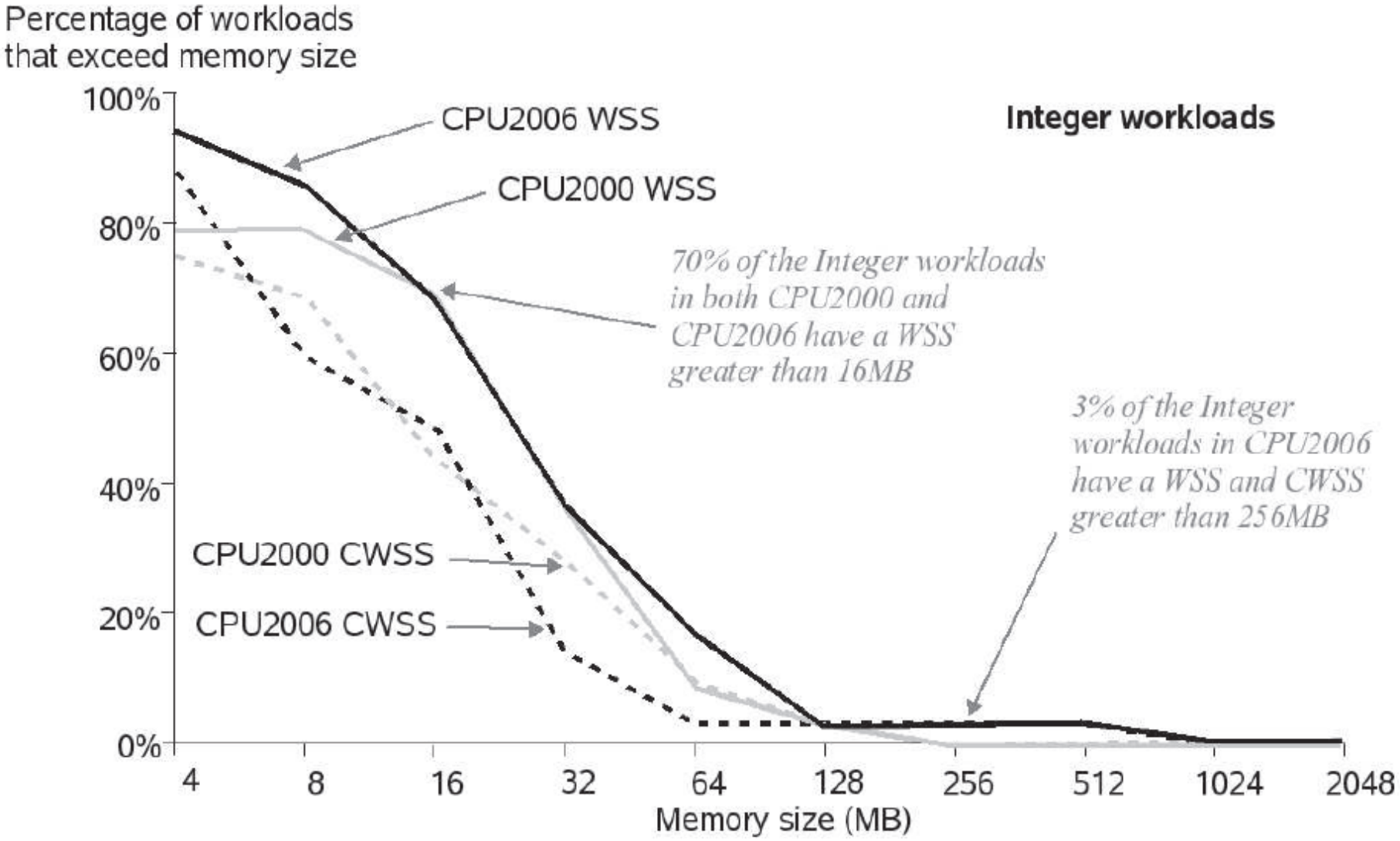

Figure 3: WSS for the Integer workloads

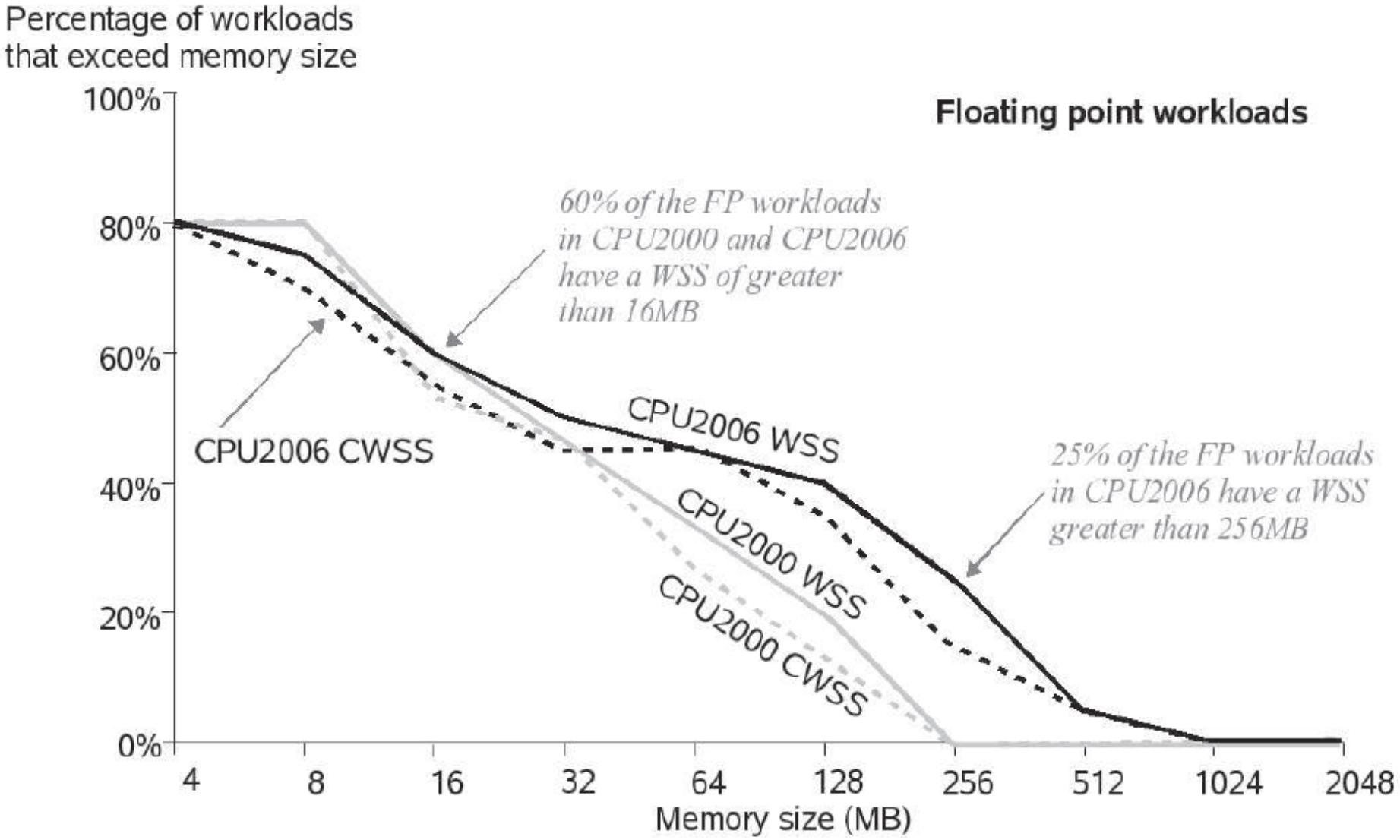

Figure 4: WSS for the Floating Point workloads

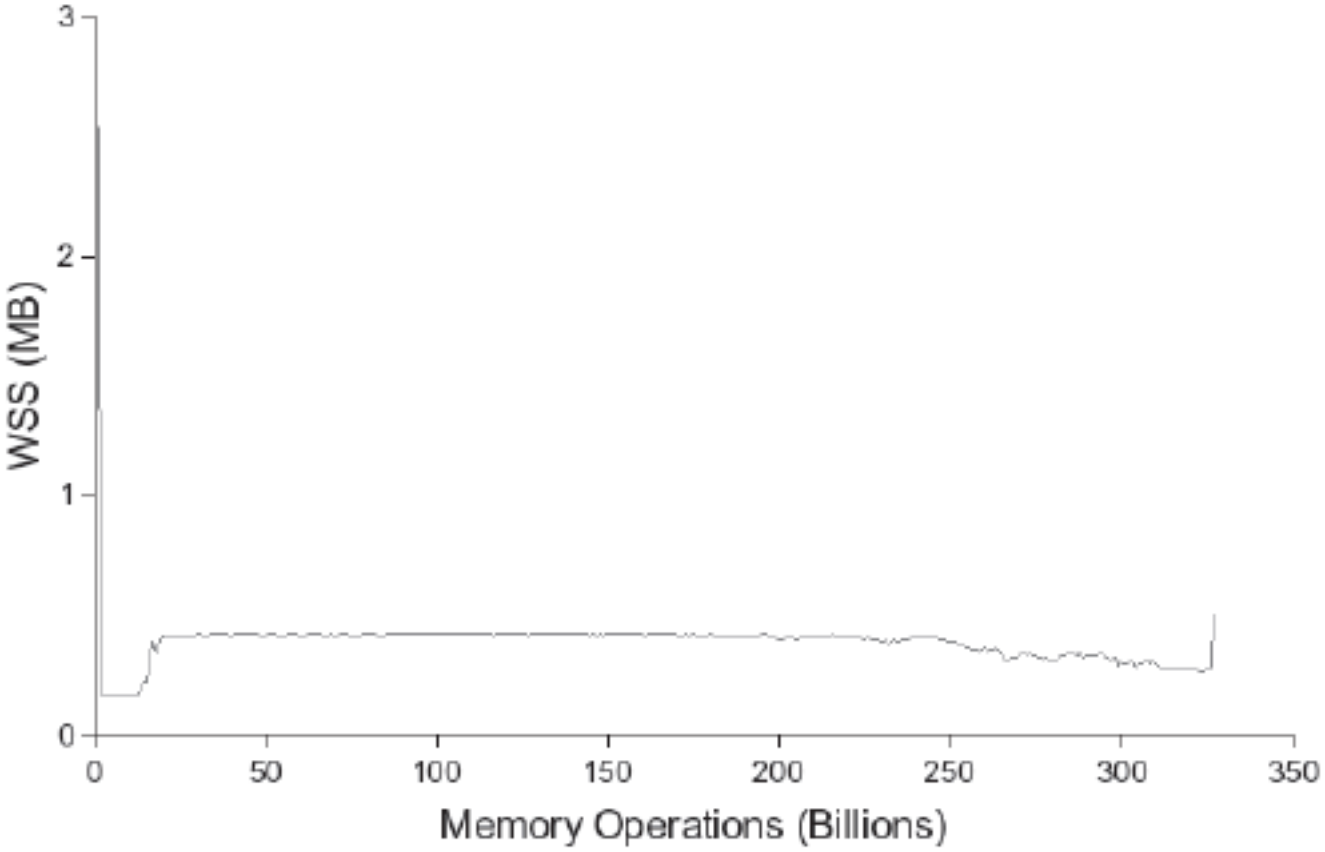

Figure 5: WSS over time of 453.povray

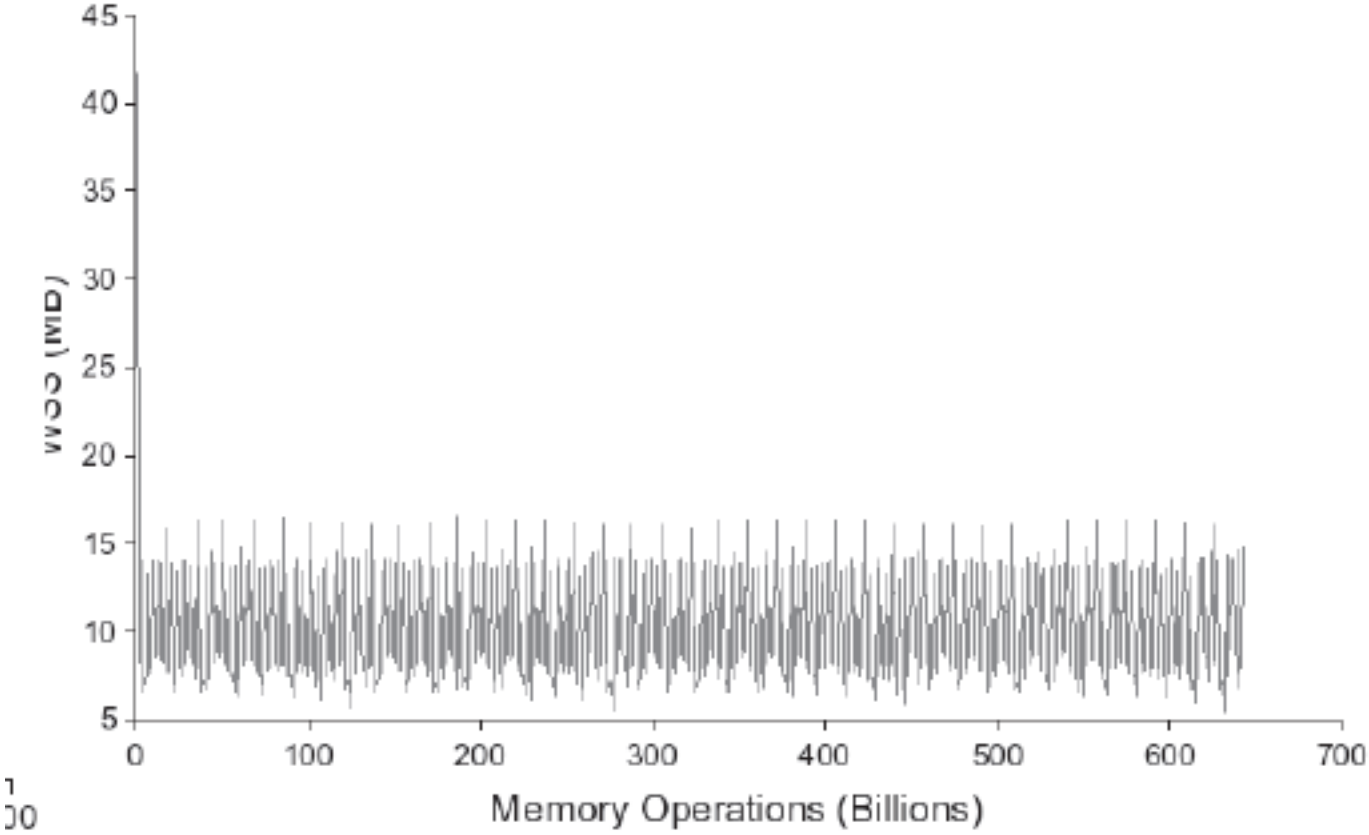

Figure 7: WSS over time for 447.namd

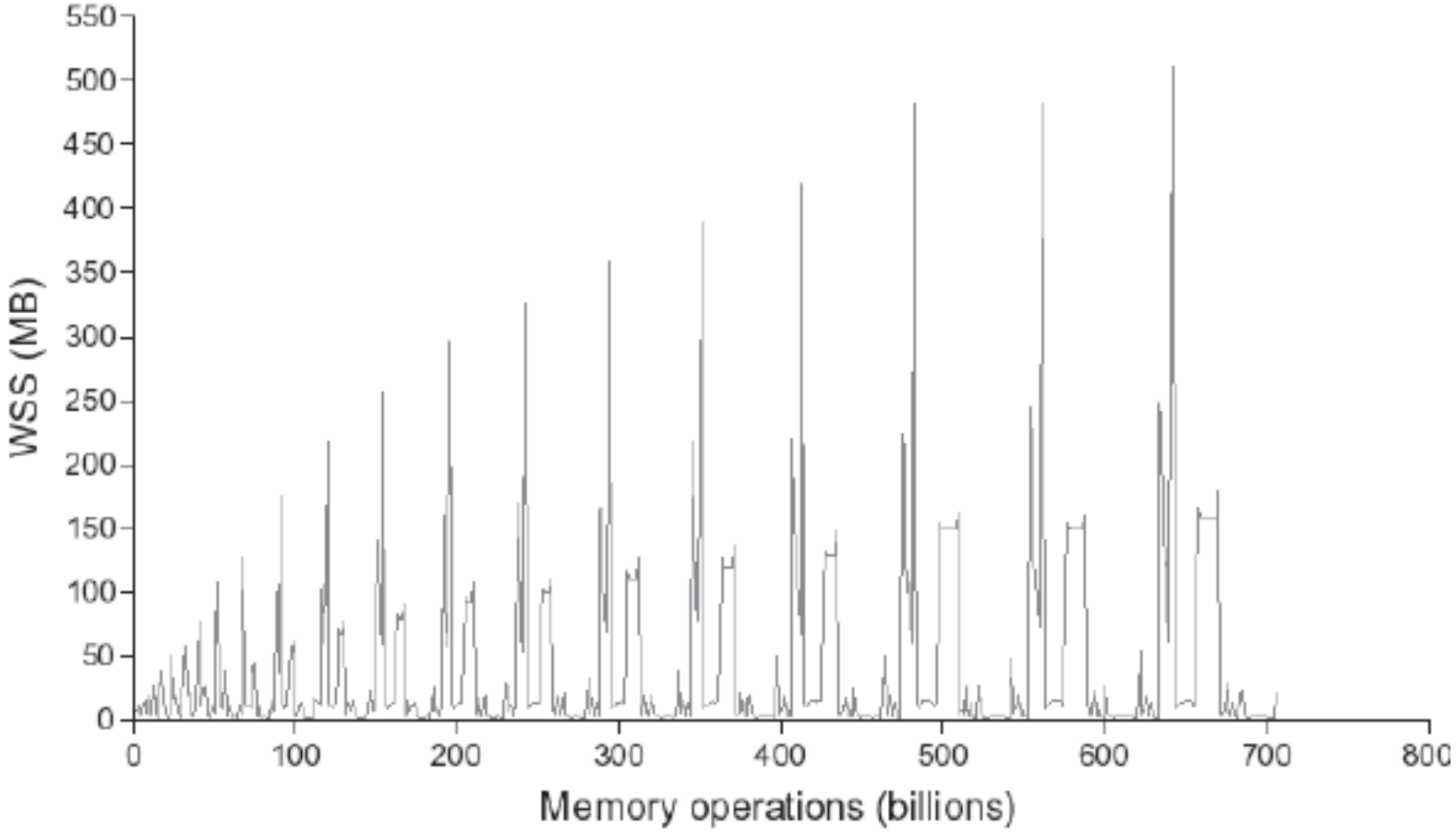

Figure 6: WSS over time for 447.dealII

Tools Used

Spot

SHADE

SHADE-based profilers

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

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
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

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

Benchmark	VSZ (MB)	RSS (MB)	WSS (MB)	Std	CWSS (MB)	Std
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
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

Benchmark	VSZ (MB)	RSS (MB)	WSS (MB)	Std	CWSS (MB)	Std
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