Introductory User's Guide to the Architect's Workbench Tools

J. Torrellas, B. Bray, K. Cuderman,
S. Goldschmidt, A. Kobrin and A. Zimmerman

Technical Report CSL-TR-88-355

May 1988

This research is supported by the Institute for Defense Analyses under contract MDA-0031-68.

Introductory User's Guide to the Architect's Workbench Tools

by

J. Torrellas, B. Bray, K. Cuderman,

S. Goldschmidt, A. Kobrin and A. Zimmerman

Technical Report CSL-TR-88-355

May 1988

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305-4055

Abstract

The Architect's Workbench is a set of simulation tools to provide insight on how the instruction set and the organization of registers and cache affect processor-memory traffic and, as a result, processor performance. This report is designed to be an introductory guide to the tools for the novice user.

Key Words and Phrases: Architecture Simulation, Cache Simulation, Register Organization, Bus Traffic.

Contents

1 Introduction and First Steps 1

2 Instruction Referencing Simulation 1

3 Data Referencing Simulation 8

A List of predefined architectures 17

B An example of Cache.conf 20

C Another example of Cache.conf 20

D File listings 21

List of Figures

Figure 1
High-level flowchart of the tools **2**

Figure 2
Detailed flowchart of the tools (1) **3**

Figure 3
Datailed flowchart of the tools (2) **4**

Plot **1**
Data cache miss rate **7**

Plot **2**
Number of misses in the instruction cache **16**

1 Introduction and First Steps

The Architect's Workbench is currently a set of tools to provide an insight on how a particular register organization and cache organization affects processor-memory traffic. It is organized as a set of program tools to be applied sequentially to a high level language source program. See figures 1, 2 and 3.

This guide will use a test program called Myfile.p (see listing 1) to proceed along the simulator tools step by step and discuss some of the results. The test program generates a series of 100 real numbers and uses bubble sort to sort them.

A set of man pages are available for on line help. To list them on the screen we use the command

awbman (name-of-page)

To print the page the command is

awbman -p (name-of-page)

For a list of tools see awbman page on awbtools.

The simulator tools are organized into two groups. One group of tools are used to simulate the instruction referencing traffic for different architectures and cache configurations and the second group simulates data referencing traffic for different architectures and cache configurations.

2 Instruction Referencing Simulation

The script file *irefsim* (see awbman page on irefsim) organizes the tools for the novice user. The tools called by *irefsim* will produce an output file for a particular architecture and a particular application showing the number of cache lines fetched, number of bits fetched, miss rate, traffic ratio and relative traffic for different cache organizations. The cache organizations consist of the line size, number of elements and associativity of different-sized caches. The caches can range from 128B to 16KB in size. The results for an application - architecture pair can be expressed relative to the no-cache memory traffic for that particular architecture - application or relative

Figure 1: High-level flowchart of the tools

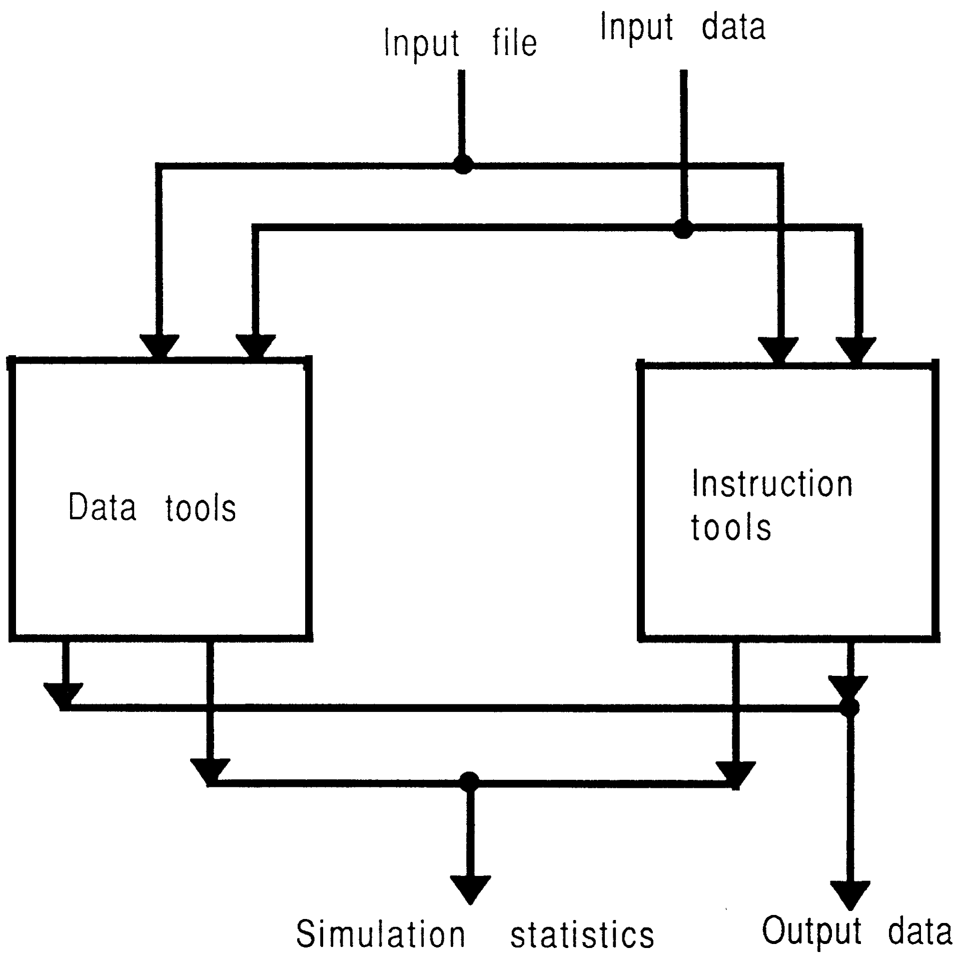


Figure 2: Detailed flowchart of the tools (1)

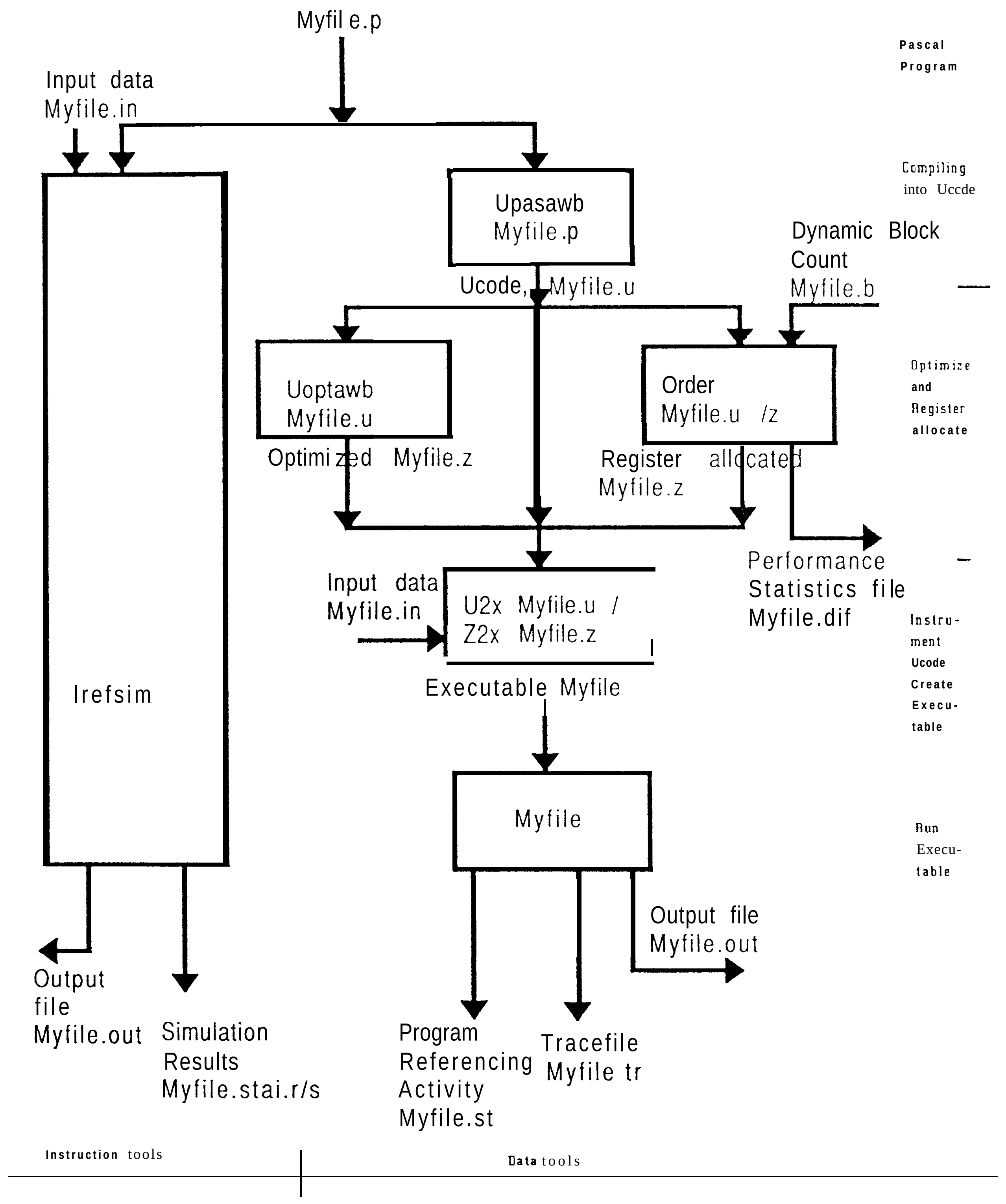
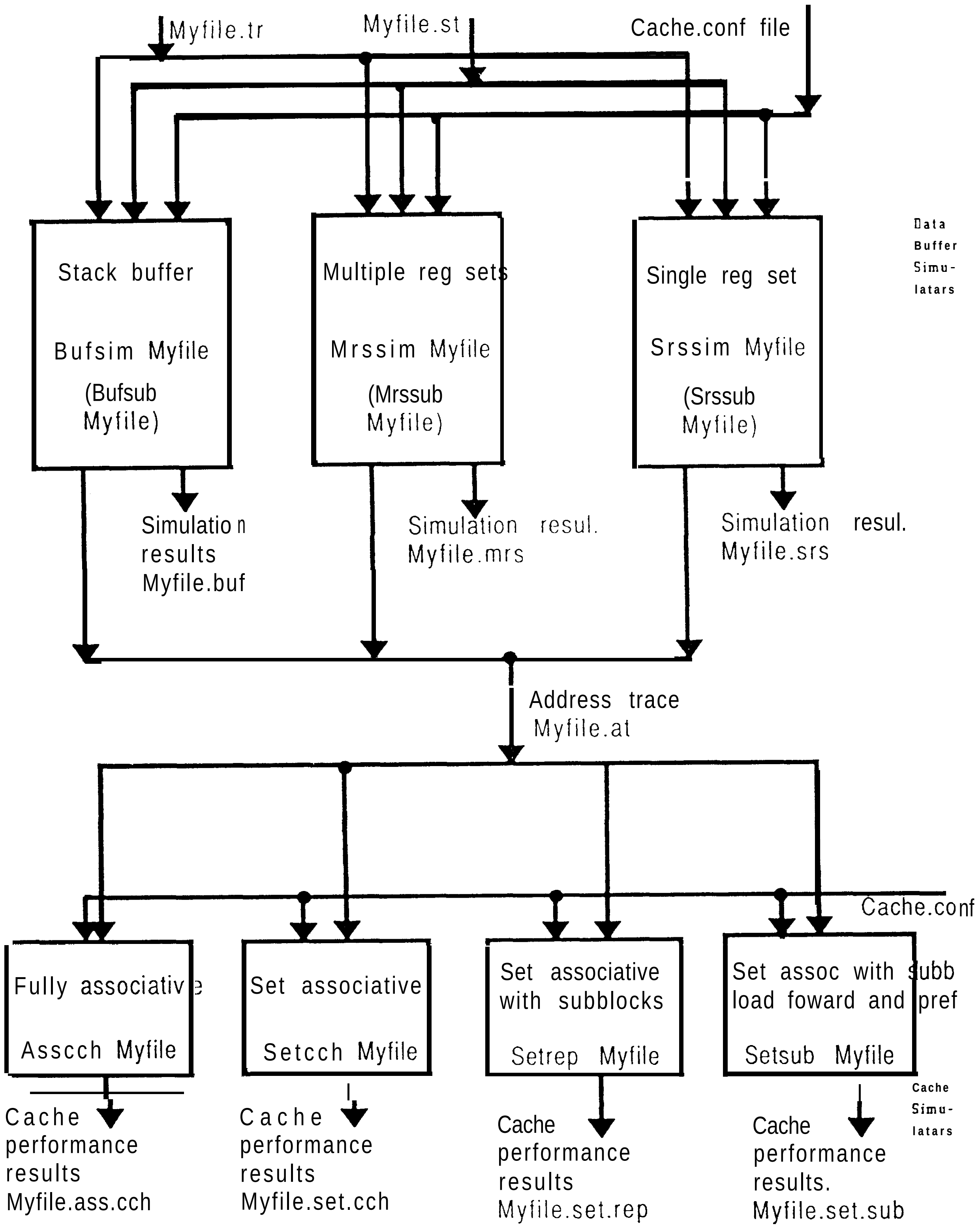


Figure 3: Detailed flowchart of the tools (2)



to the no-cache memory traffic for a different architecture with the same application.

Irefsim takes the following arguments:

irefsim {architecture} {application} { smcache ||*regcache*} { *reference_arch*}

Architecture refers to the type of architecture we want to simulate. Architectures are divided into register and stack machines. Parameters that can be specified include number of bits per functional instruction, number of additional bits per memory reference, alignment policy for branch targets, etc. The list of architectures supported by *irefsim* and the parameters that specify them are in Appendix A. They include IBM/370 style architectures, RISC, stack, etc. The user can simulate new architectures by including their parameters in the file archdir/more.parameters in the current version of irefsim, in a style consistent with that found in /awb/interface/parameters. Awbman pages on *Regstat*, awb.*Stackstat* and *Dcastat* give information on the parameters used by register, stack and direct correspondence machines respectively. These programs are directly called by *irefsim*.

Application is the application, in our case Myfile. Irefsim expects a full directory structure named benchdir/Myfile/Myfile.p. *(See irefsim).*

Smcache and *regcache* are two different programs that simulate different size caches. *Smcache* simulates caches with size ranging from 128B to 4KB and line sizes of 32/64/128/256 bits. *Regcache* considers caches from 512B to 16KB and line sizes of 64/128bits.

Reference-archit is the architecture that is going to be used as a base in our study. That is, all the results of *Architecture* will be made relative to the performance of *Reference-archit.* If the user wants to study an architecture relative to its own no-cache traffic, *Reference-archit* has to be substituted by the keyword *self.* To use an architecture as a reference architecture it first has to be run in the self mode, to create the reference files. *Irefsim* expects every architecture to have a directory residing in archdir.

The files *Irefsim* needs as input in benchdir/Myfile are:

Myfile.p: the source code.

Myfile.in: the input to the source program (this file may be empty).

The output files of *irefsim* are the following:

> Archdir/Myfile.sti: file that contains static block usage of the application. This file is a mapping of each basic block to references to the instructions this basic block contains, the block offset and the block length. Since different architectures have different instructions this file is architecture dependent.
>
> Archdir/Myfile.sta: file that contains the simulation results. This file is dependent on the architecture and the application. If it is created by *smcache, irefsim* renames it to Myfile.stai.s, otherwise it is renamed to Myfile.stai.r.
>
> Benchdir/Myfile.out: physical output of the program. In our example this file would contain the list of ordered numbers.
>
> Benchdir/Myfile.b: file that contains a list numbering each basic block of the application and its dynamic use. This file is independent of the architecture and is only dependent on the application. It is only created if it is required to preform the specified register allocation.

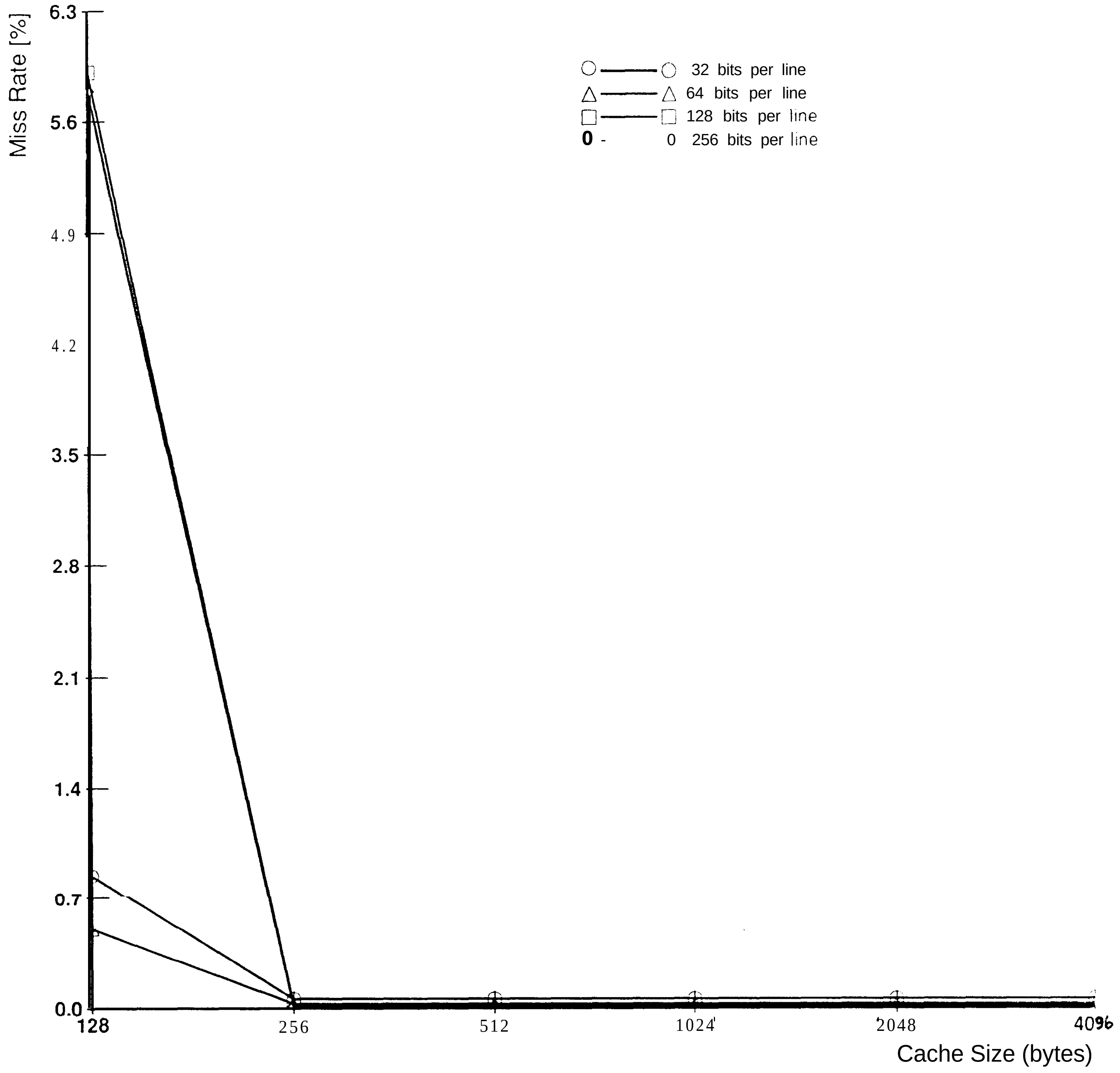
To simulate a machine with an IBM-370 like instruction set with the specific switches given in appendix A under IBM360-2, and for caches in the range 128B to 4KB, use the command

```
irefsim IBM360-2 Myfile smcache self
```

The result is Myfile.stai.s which is shown in listing 2. The file gives us the number of instructions read from the cache (1), plus the lines fetched and miss ratio for different cache sizes and different line sizes. For example (2), for a direct mapped I-cache with 64 bit lines and a total size of 128 bytes, a total of 878 lines would be fetched from memory. To get plots of the miss rates and traffic ratios type the command

```
awbplot archdir/Myfile.stai.s
```

Plot 1. Data cache miss rate



IBM360-2 Direct Mapped Cache Miss Rate (myfile)

If your system is running Xwindows 10.4 you can view the plots. You also have the option of having a postcript file of the plot being sent to the default printer . Plot 1 shows a plot of the miss rates for direct mapped caches as expressed in the file Myfile.st ai.s.

If we had specified regcache then irefsim would have produced the file Myfile.stai.r (listing 3), which contains the same information as in the case of listing 2 but for different range of caches.

The relative traffic data in listings 2 and 3 is the traffic of the IBM360-2 with respect to its own no cache traffic. Once we have the results for the IBM360-2 (our reference architecture) ***we*** can use ***irefsim*** to give us the traffic of another architecture relative to the IBM360-2 architecture no-cache traffic (for the application Myfile). The architecture we will compare to IBM360-2 is one with a 32 bit fixed sized instruction set called fix32-0. This architecture is encoded in a way similar to the RISC computers. We use the command:

```
irefsim fix32-0 Myfile smcache IBM360-2
```

and the result is listing 4. The number of lines fetched, bits fetched, miss rate and traffic ratio are absolute values, the same as the values found running fix32-0 in the self mode as shown in listing 5. The column labeled Relative Traffic represents data relative to the IBM360-2 without cache. Some things that can be observed are that e.g. for 64 bit lines, the IBM360-2 references 48 different lines, while the fix32-0 references 80. As a result, for small caches, e.g. 128 bytes, the IBM360-2 without cache performs better than the fix32-0 with a cache.

3 Data Referencing Simulation

The basic sequence of operations that has to be done to produce data reference statistics files is as follows:

1. The first step is to compile the source program into an intermediate language representation called U-code. The U-code is a standard representation to allow portability to different machine languages.

2. Do a register allocation and possibly a global optimization of the U-code.

3. Instrument the code, assemble and run the program. A trace file Myfile.tr will be generated.

4. Use Myfile.tr to simulate a type of data buffering architecture. Available architectures, with the tools that are used to simulate them in brackets, are: stack buffer architecture [Bufsim] , multiple register set (register windows) buffer architecture [Mrssim] and single register set architecture [Srssim]. The results are a simulation statistics file and a file with address trace to memory system, Myfile.at. All the mentioned tools take the same input files.

5. Use any of the above Myfile.at files and a cache configuration file (cache.conf) to simulate different types of caches. The available types are: full associative caches with LRU [Asscch]; 1/2/4 way set associative caches with LRU/ random / FIFO replacement [Setcch] ; set associative caches with LRU replacement, with subblocks, load fowarding and prefetching [Set sub]; set associative caches with LRU / random / FIFO replacement [Setrep]. These programs all take the same input files.

Let us consider each of these steps in order. The reader is reminded that all these operations expect the input files in the current directory and produce output files in the current directory.

1. To compile the source program into U-code we use *Upasawb*. This pass takes Myfile.p and produces Myfile.u, the U-code file.

   A typical call would be of the form:

   ```
   upasawb [options] Myfile.p
   ```

   For available options see the awbman page on upasawb. In our example we used the default settings.

2. Two tools can be used to do register allocation: *Uoptawb* and *Order*.

Uoptawb does global optimization of the U-code files. This includes global register allocation, detection of redundant stores, code motion, etc. The command is of the form:

```
uoptawb [options] Myfile.u
```

and produces the output in file Myfile.z. For a complete set of the options see awbman page on uoptawb. In our example we used the default settings.

The Order register allocator is the other alternative. See awbman page on order for information. Unless the flag *-us:3* is given, *order* needs as one of its inputs Myfile.b, a file containing Myfile's dynamic basic block counts, which is currently different to Myfile.b used in the instruction referencing simulators. This file is characteristic of the application, not dependent on the architecture/cache type simulated and gives us a listing of basic block number and number of times it was used. To generate Myfile.b we will have to use some of the tools in section 3 above.

The steps to follow to get Myfile.b start by producing an executable with *u2x*, as in

```
u2x Myfile
```

this command will create the executable Myfile and a file called Myfile.st which contains a static summary of the high-level language program referencing activity. We run Myfile

```
Myfile < Myfile.in > Myfile.out
```

and the result is the trace file Myfile.tr which reports the order of execution of the basic blocks and the addresses for indirect references.

To produce Myfile. b we run any of the architecture simulators mentioned above using the -bcnt switch. This process can be done with any of them because it is architecture independent. The command is:

```
xxxsim -bcnt Myfile
```

where xxx refers to *srs*, *mrs* or *buf*.

These tools, covered later on, expect to have three files, Myfile.st, Myfile.tr and a cache configuration file cache.conf. Cache.conf is not used at this point, it may be empty, but the file must exist.

Since Myfile.b is present, *Order* can now be invoked (with a -us flag other than *3). Order* maps locals in registers for the entire time a procedure is active and so all registers have to be saved on procedure call/return. The command is

```
Order [switches] Myfile.ext
```

where ext is some extension. Be sure not to give a file Myfile.z for input. For the purposes of this example we will use the extension u, corresponding to the u-code file. It will take Myfile.u and Myfile.b, producing the variable reordered U-code file Myfile.z. It also generates a file with the count usage of each block Myfile.dif if the correct switches are given. The count usage will be a static count if we included among the switches -us:2, otherwise the default is a dynamic count. The -1 flag tells which types of variables should be reordered. If the -1 flag is omitted then no reordering occurs. For example we used the following command to reorder all variables:

```
Order -l:7 -us:1 Myfile.u.
```

3. To generate the trace file we have to instrument the code, assemble and run the program. We instrument the code and produce the executable with the script x2x. This script runs the same programs as the *u2x* script, but uses for input Myfile.z instead of Myfile.u.

   ```
   z2x Myfile
   ```

   which produces Myfile and Myfile.st. We then run the executable

   ```
   Myfile < Myfile.in > Myfile.out
   ```

   to get the trace file Myfile.tr.

4. The following tools simulate different on chip data buffering organizations and optionally a first level cache.

   The tools accept as input files the files generated above, namely Myfile.tr and Myfile.st. In addition, the buffer simulators need the file cache.conf, specifying some buffer characteristics and the configuration of the first level cache. Running the xxxsims will generate one or two files. The file that is always generated contains simulation statistics. It is called different names depending on the generating tool: Myfile.buf if it is generated by *bufsim;* Myfile.mrs if generated by mrssim; and Myfile.srs if generated by *srssim*. The second file contains address traces to the memory system. It is called Myfile.at and is only created if the processor configuration specified in the cache.conf file contains an off chip cache description, which is the case when we want to use the tools described in the next section.

   Cache.conf is a file that specifies the type of cache memory configuration we want to simulate. In the appendices B and C we give a few examples of cache.conf files we can use (see awbman page on cacheconf).

   For the data buffer simulators it is assumed that all registers which buffer data traffic are general purpose registers. This does not necessarily mean that all registers need to be general purpose. All the data buffer simulators need eight additional registers to keep the evaluation stack (2), static information (3) and constants and so forth (3). Some of these registers could be special purpose registers.

   If we had a 16 register set machine, 8 registers would be used by the data buffer's register set architecture, leaving eight registers available for allocation strategies. As an example, let us allocate six registers for buffering local (stack) variables (switch -lrs:6) and two registers for buffering global variables (switch -grs:2). Globals are normally in the stack so we must map them into a separate global space (switch -gs:3 global simple and structure variables). We are going to use the cache.conf file in appendix B with the trace file generated by Order with the above mentioned switches.

   ```
   srssim -grs:2 -gs:3 -lrs:6 Myfile
   ```

We get the statistics file Myfile.srs (listing 6) and the memory system address trace file Myfile .at .

Myfile.srs gives us the number of global traffic to cpu: reads and writes, the local traffic to cpu: reads and writes and other information. The reader can see the number of 32b word references made by the ALU in (1) and the total number of word references that are made by the ALU and registers to the cache or to the memory system if there is no cache in (2). See awbman page on srssim for the list of switches srssim offers.

Mrssim provides for simulating architectures with multiple register sets in a register window fashion (see awbman page on mrssim). Let us assume we still have the same ISA (instruction set architecture) as before which limits the number of visible registers to 16 but we want to see the benefit of register windows. The 16 registers have eight substracted because of the register set architecture of the data buffer simulators, leaving eight visible registers. If we want to make six visible registers windowed (leaving two registers for buffering global variables), we can for example allocate four registers for locals and one register for each parameter section. The number of registers which are visible by a window is the number of in-parameter registers plus locals plus out-parameter registers, that is, the number of locals plus two times the number of parameter registers. In our example this value is six.

We will run the same cache configuration as before simulating 8 different sized windows (switch -cnt:8) and wanting the memory address trace to come from the buffer with 4 windows (switch -win:4). The parameter size is 2 registers (switch -psize:2) and the local size is 4 registers, making the total size of the window 6 registers (switch -wsize:6).

```
mrssim -gs:3 -grs:2 -win:4 -cnt:8 -wsize:6 -psize:2 Myfile
```

This command produces a file with the results of the simulation called Myfile.mrs (listing 7) and a file with an address trace to memory system Myfile.at. In file Myfile.mrs we see that the number of ALU references is the same and that the number of references that leave the processor is still high with 4 windows. These references are global references and the small amount of global registers the processor have are not able to intercept them.

To simulate stack buffers (e.g. AT&T's CRISP) *we* use *Buffsim*.

```
bufsim [switches] Myfile
```

which would expect Myfile.st, Myfile.tr and cache.conf. It would produce Myfile.buf, a listing of the simulation results, and if suitable, Myfile.at, an address trace to memory system. Among the available switches (see awbman page on bufsim) we can specify the number of buffers to simulate (where the buffer sizes start from an initial size and each buffer is twice as large as the previous one), the size of the first buffer, the save/restore policy, etc.

In addition to the above three tools we have srssub, mrssub and bufsub. They differ to the presented tools in that their first level caches have subblocks and so they take slightly different switches. They produce the same output files as their corresponding tool above. (see awbman page on srssub, mrssub and bufsub).

5. Once Myfile.at has been created we can simulate different cache types. Notice that the type of cache we simulate here may be a second level cache (if Myfile.at was created using a first level cache) or may be a first level cache (if Myfile.at was created without any first level cache).

   There are four tools which take Myfile.at generated above and file cache.conf and produce a cache performance file. The tools, their object and the name of the output file are:

   Asscch Simulates fully associative caches with LRU. Output file: Myfile.as.cch.

   Setcch Simulates 1/2/4 way set associative caches with LRU/ random/ FIFO replacement strategy. Output file: Myfile.set.cch.

   Setsub Simulates set associative caches with LRU replacement with subblocks, load fowarding and prefetching. Output file: Myfile.set .sub.

Setrep Simulates set associative caches with LRU/random/FIFO replacement with subblocks. Output file: Myfile.set.rep.

In all of them 7 types of data references are distinguished: static, global, stack, structure, heap, frame and i/o (i/o references are currently not generated). The output file reports the counts and miss ratio for the separate and combined references for several cache sizes.

The switches available in *asscch* allow the user to specify the maximum number of cache blocks to simulate, the cache block size, etc. See awbman page on asscch. The output file resulting from using default values, the cache.conf in appendix B and the trace file Myfile.at resulting from srssim is given in listing 8. The actual command typed in is

```
asscch Myfile
```

In listing 8 we are given the simulation results for various sized caches, from 32 to 32K bytes, all with a 32 byte line size. Fot each of these caches we see the number of read misses, write misses, total misses, the read miss ratio, write miss ratio and total miss ratio. Other information includes traffic, etc. The file shows that in a cache with 128 bytes the traffic to memory is 6560 reads and 5848 writes (all in 32b words and highlighted in (1)). If the cache were organized in a write through way, the traffic to memory would be 6560 reads and 7455 writes.

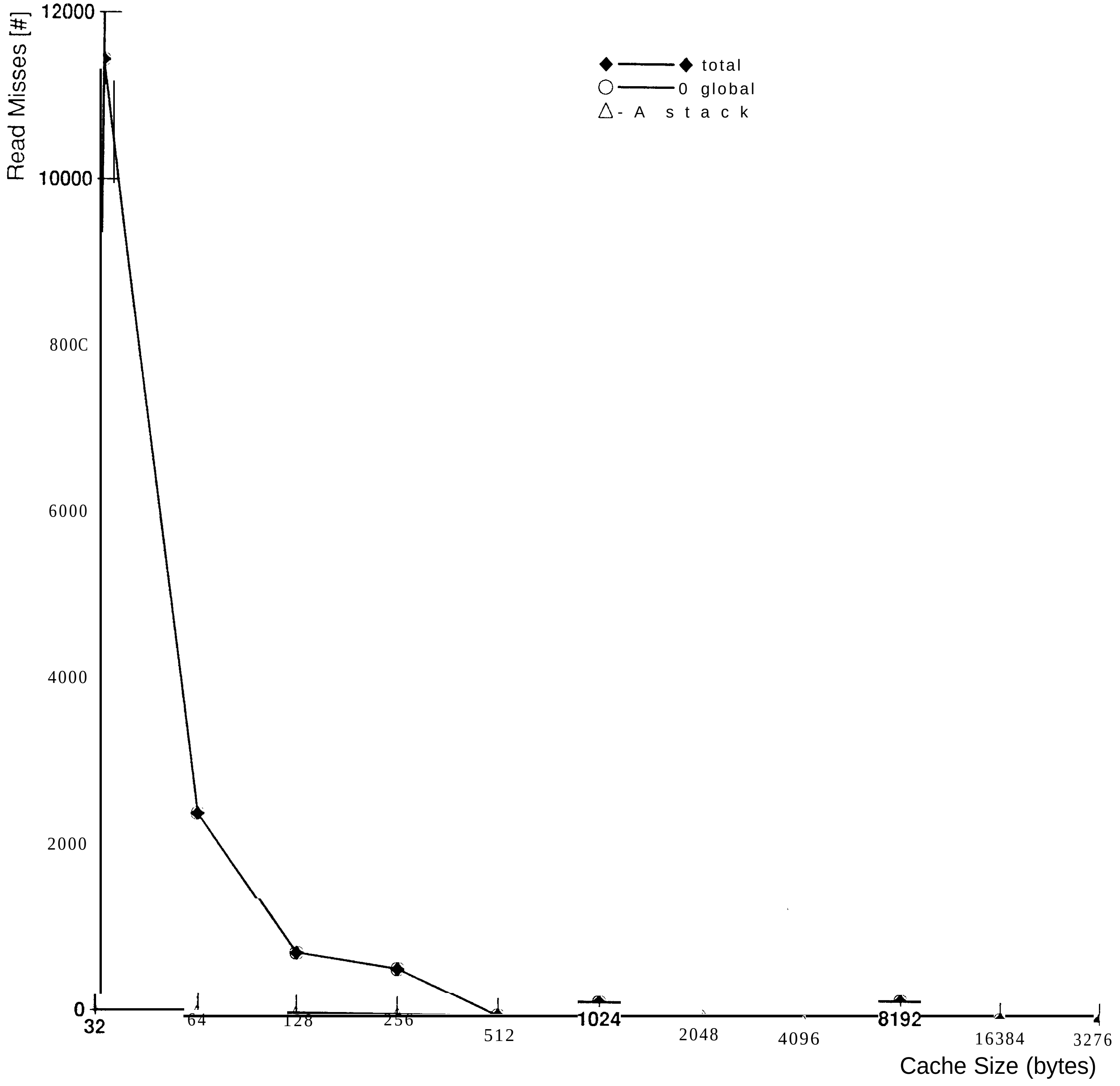
To get plots of the misses or miss ratios, type the command

```
awbplot Myfile.as.cch
```

If your system is running Xwindows 10.4 you can view the plots. You also have the option of having a postcript file of the plot being sent to the default printer. Plot 2 shows a plot of the read misses as expressed in the file Myfile.as.cch.

We now use setrep (see awbman page on setrep) to simulate set associative caches with subblocks. To be able to simulate subblocks we will change our cache.conf file to the one in appendix C, in which we simulate a cache with line size of 32 bytes and 4 byte sublocks. Let

Plot 2: Number of misses in the instruction cache



Basic Fully Associative Cache (myfile)

us suppose we want a foru way set associative cache using a FIFO replacement strategy. Note we do not have to create a new Myfile.at.

We use the command

```
setrep -a:4 -r:2 Myfile
```

The resulting file, Myfile.set.rep is listed in listing 9. The file also gives us the number of misses in (1) and the traffic to memory in *(2)* for a 128 byte cache. From the numbers we see that thanks to the increase in the cache associativity, the read and write traffic have been reduced. The number of dirty lines is now 3921 as opposed to 73 1 before.

The available switches in *setsub* allow us to determine the associativity, the cache block size, number of caches to simulate, whether we have load fowarding or not, whether we have an invalidation of the line on write through, the number of subblocks prefetched. etc. See awbman page on setsub.

The options in *setcch* let us choose the number of cache blocks, the cache block size, the replacement algorithm, etc. Please see awbman page on setcch.

The running of *setsub* and *setrep* would proceed in the same way as the previous two examples.

A List of predefined architectures

The architectures recognized by the simulator use the following conventions. We then list the predefined architectures.

”reg” means register machine.

“stack” means stack machine.

”o” gives the number of bits for simple RR (register transfer) instruction.

"m" gives t he number of additional bits for each memory reference.

"c" gives the number of addit ional bits for small constant embedded.

"a" gives the alignment of branch targets.

"r" gives the number of registers available for temporaries.

"1" gives the number of registers available for locals.

"p" gives the number of operand sources allowed from memory (0,1,2) (default 1).

"s" means that op destination allowed in memory (default false).

"w" means assume ideal register set windows (default false).

"b" gives the number of additional bits for each branch target reference.

"e" means that it uses exact register allocation.

"t" means that the architecture only allows two registers in an RR instruction (default is false).

"x" means that the source register and the destination register in an RX instruction format are the same (default false).

Some other switches allow the user to specify the instruction set constant handling, etc. For a complete set of switches please see the awbman page on *regstat* for register machines and *stackstat* for stack machines.

regref	reg	-	-	-	-		-	-	-	-	-
B-mmm	reg	r:5	l:5	o:8	m:24	b:24	c:8	a:8	p:2	s	e
B-mmr	reg	r:5	l:5	o:8	m:24	b:24	c:8	a:8	p:2	-	e
B-rmr	reg	r:5	l:5	o:8	m:24	b:24	c:8	a:8	p:l	-	e
B-rrr	reg	r:5	l:5	o:8	m:24	b:24	c:8	a:8	p:o	-	e
IBM360-1	reg	r:1	-	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360-2	reg	r:2	-	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360-3	reg	r:3	-	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360-4	reg	r:4	-	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360-5	reg	r:5	-	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360l-1	reg	r:5	l:1	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360l-2	reg	r:5	l:2	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360l-3	reg	r:5	l:3	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360l-4	reg	r:5	l:4	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360l-5	reg	r:5	l:5	o:16	m:16	b:16	c:0	a:16	-	-	-
IBM360o-1	reg	r:5	l:1	o:16	m:16	b:16	c:0	a:16	-	-	e
IBM360o-2	reg	r:5	l:2	o:16	m:16	b:16	c:0	a:16	-	-	e
IBM360o-3	reg	r:5	l:3	o:16	m:16	b:16	c:0	a:16	-	-	e
IBM360o-4	reg	r:5	l:4	o:16	m:16	b:16	c:0	a:16	-	-	e
IBM360o-5	reg	r:5	l:5	o:16	m:16	b:16	c:0	a:16	-	-	e
IBM360o-6	reg	r:5	l:6	o:16	m:16	b:16	c:0	a:16	-	-	e
IBM360o-7	reg	r:5	l:7	o:16	m:16	b:16	c:0	a:16	-	-	e
IBM360o-8	reg	r:5	l:8	o:16	m:16	b:16	c:0	a:16	-	-	e
IBM360o-9	reg	r:5	l:9	o:16	m:16	b:16	c:0	a:16	-	-	e
IBM360o-w	reg	r:5	l:15	o:16	m:16	b:16	c:0	**a:16**	-	w	e
b6700	stack	r:16	-	o:8	-	b	-	~~a:8~~	~~-~~	-	-
b6700a	stack	r:16	-	o:8		b	-			-	-
fix32-0	reg	r:7	l:0	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-1	reg	r:7	l:1	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-2	reg	r:7	l:2	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-3	reg	r:7	l:3	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-4	reg	r:7	l:4	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-5	reg	r:7	l:5	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-6	reg	r:7	l:6	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-7	reg	r:7	l:7	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-8	reg	r:7	l:8	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-9	reg	r:7	l:9	o:32	m:0	b:0	c:0	a:32	p:0	-	e
fix32-m	reg	r:7	l:5	o:32	m:0	b:0	c:0	a:32	p:l	-	e
fix32-w	reg	r:7	l:15	o:32	m:0	b:0	c:0	a:32	p:0	-	e
s816a128	stack	-	-	o:8		-	c:16	a:128	-	-	-
s816a64	stack	-	-	o:8		-	c:16	a:64	-	-	-
stack	stack	-	-	-		-		-	-	-	-
stack816	stack	-	-	o:8		-	c:16	a:8	-	-	-
stack816a	stack	-	-	o:8		-	c:16	-	-	-	-

For more information on the architecture classification and switches, please see the paper *The Effects of Processor Architecture on Memory Traffic* by C. Mitchell and M. Flynn.

B An example of Cache.conf

This file specifies the configuration of a cache. The first level buffer has a datapath to/from cache of 4B (default). The cache has a datapath to/from memory of 4B, blocks are 32 bytes, and writes are cached. The result file will contain data for different sized caches. The cache does allocate on write misses and has no subblocks in a line. We count both regular and maintenance accesses. The latter are procedure calls and returns, stack link accesses, etc. Both read and writes to cache include static references, global references, stack references, structure references, heap and frame references, all of which are cached in a copy-back fashion.

```
2 3 4 4 32 1024 0 1 0 0 1

1 1

1 1 1 1 1 1 0 0

1 1 1 1 1 1 0 0
```

C Another example of Cache.conf

The same configuration as before but we support subblocks of size 4 bytes in a cache where the line size is 32 bytes.

```
2 3 4 4 32 1024 0 1 1 4 1

1 1

1 1 1 1 1 1 0 0

1 1 1 1 1 1 0 0
```

D File listings

```
program myfile(output);
const    LEN_ARRAY= 100;
const    FACTOR = 0.231;

var arr:array[1..LEN_ARRAY] of real;
    k,i,j,temp: integer;


begin
      for j  := 1 to 20 do begin
arr[j] := j * 34 ;
arr[j+20 ]:= j * 24 ;
arr[j+40 ]:= j * 14 ;
arr[j+60 ]:= j * 4 ;
arr[j+80 ]:= j * I ;
      end;
   for i  := 1 to 100 do
     for j  := 1 to 100 - i do
         if arr[j] > arr[j+1] then begin
temp  := arr[j+1];
arr[j+1]  := arr[j];
arr[j]  := temp
end;

   for i  := 1 to 100 do
   begin
       write(arr[i]);
       if i mod 5 = 0 then begin
          nriteln
       end;
   end;

end.
```

Listing 1

```
*** Cache performance statistics ***   Benchmark: myfile
Instructions read from cache:  175212   (4054592 bits   506824 bytes)    --(1)
Basic blocks traced: 14069, Relative Ho-Cache Traffic:        1.0000

LineSize===> 32 bits
Total lines in memory: 104,   Distinct lines referenced (approx) = 97
Estim. Optimal Hiss Rate & Traffic Ratio:          0.06          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct Mapped
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
```

```
  4096          105        3360      0.06%       0.08%        0.0008
  2048          105        3360      0.06%       0.08%        0.0008
  1024          105        3360      0.06%       0.08%        0.0008
   512          105        3360      0.06%       0.08%        0.0008
   256          105        3360      0.06%       0.08%        0.0008
   128         1483       47456      0.85%       1.17%        0.0117

Two Way Associative
Cache Size    Lines        Bits       Miss     Traffic      Relative
(bytes)      Fetched    Fetched       Rate       Ratio       Traffic
  4096          105        3360      0.06%       0.08%        0.0008
  2048          105        3360      0.06%       0.08%        0.0008
  1024          105        3360      0.06%       0.08%        0.0008
   512          105        3360      0.06%       0.08%        0.0008
   256          105        3360      0.06%       0.08%        0.0008
   128         1534       49088      0.88%       1.21%        0.0121

LineSize===> 64 bits
Total lines in memory: 52,   Distinct lines referenced (approx) = 48
Estim. Optimal Miss Rate & Traffic Ratio:          0.03          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct Mapped
Cache Size    Lines        Bits       Miss     Traffic      Relative
(bytes)      Fetched    Fetched       Rate       Ratio       Traffic
  4096           53        3392      0.03%       0.08%        0.0008
  2048           53        3392      0.03%       0.08%        0.0008
  1024           53        3392      0.03%       0.08%        0.0008
   512           53        3392      0.03%       0.08%        0.0008
   256           53        3392      0.03%       0.08%        0.0008
   128          878       56192      0.50%       1.39%        0.0139      --(2)

Two Way Associative
Cache Size    Lines        Bits       Miss     Traffic      Relative
(bytes)      Fetched    Fetched       Rate       Ratio       Traffic
  4096           53        3392      0.03%       0.08%        0.0008
  2048           53        3392      0.03%       0.08%        0.0008
  1024           53        3392      0.03%       0.08%        0.0008
   512           53        3392      0.03%       0.08%        0.0008
   256           53        3392      0.03%       0.08%        0.0008
   128          900       57600      0.51%       1.42%        0.0142

LineSize===> 128 bits
Total lines in memory: 26,   Distinct lines referenced (approx) = 24
Estim. Optimal Miss Rate & Traffic Ratio:          0.01          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct Mapped
Cache Size    Lines        Bits       Miss     Traffic      Relative
(bytes)      Fetched    Fetched       Rate       Ratio       Traffic
  4096           27        3456      0.02%       0.09%        0.0009
  2048           27        3456      0.02%       0.09%        0.0009
  1024           27        3456      0.02%       0.09%        0.0009
   512           27        3456      0.02%       0.09%        0.0009
   256           27        3456      0.02%       0.09%        0.0009
   128        10358     1325824      5.91%      32.70%        0.3270
```

```
Two Way Associative
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
  4096           27       3456    0.02%      0.09%       0.0009
  2048           27       3456    0.02%      0.09%       0.0009
  1024           27       3456    0.02%      0.09%       0.0009
   512           27       3456    0.02%      0.09%       0.0009
   256           27       3456    0.02%      0.09%       0.0009
   128        11053    1414784    6.31%     34.89%       0.3489

LineSize===> 256 bits
Total lines in memory: 13,   Distinct lines referenced (approx) = 12
Estim. Optimal Miss Rate & Traffic Ratio:          0.01          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct Mapped
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
  4096           14       3584    0.01%      0.09%       0.0009
  2048           14       3584    0.01%      0.09%       0.0009
  1024           14       3584    0.01%      0.09%       0.0009
   512           14       3584    0.01%      0.09%       0.0009
   256           14       3584    0.01%      0.09%       0.0009
   128        10148    2597888    5.79%     64.07%       0.6407

Two Way Associative
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
  4096           14       3584    0.01%      0.09%       0.0009
  2048           14       3584    0.01%      0.09%       0.0009
  1024           14       3584    0.01%      0.09%       0.0009
   512           14       3584    0.01%      0.09%       0.0009
   256           14       3584    0.01%      0.09%       0.0009
   128        10939    2800384    6.24%     69.07%       0.6907

Architecture: IBM360-2 Reference architecture: self
```

Listing 2

```
*** Cache performance statistics ***   Benchmark: myfile
Instructions read from cache:  175212   (4054592 bits   506824 bytes)
Basic blocks traced: 14069, Relative No-Cache Traffic:        1.0000

LineSize===> 64 bits
Total lines in memory: 52,   Distinct lines referenced (approx) = 49
Estim. Optimal Miss Rate & Traffic Ratio:          0.03          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct Mapped
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
```

```
 16384           53      3392     0.03%       0.08%        0.0008
  8192           53      3392     0.03%       0.08%        0.0008
  4096           53      3392     0.03%       0.08%        0.0008
  2048           53      3392     0.03%       0.08%        0.0008
  1024           53      3392     0.03%       0.08%        0.0008
   512           53      3392     0.03%       0.08%        0.0008

Two Yay Associative
Cache Size    Lines      Bits       Miss    Traffic      Relative
(bytes)      Fetched   Fetched      Rate      Ratio       Traffic
 16384           53      3392     0.03%       0.08%        0.0008
  8192           53      3392     0.03%       0.08%        0.0008
  4096           53      3392     0.03%       0.08%        0.0008
  2048           53      3392     0.03%       0.08%        0.0008
  1024           53      3392     0.03%       0.08%        0.0008
   512           53      3392     0.03%       0.08%        0.0008

LineSize===> 128 bits
Total lines in memory: 26,   Distinct lines referenced (approx) = 24
Estim. Optimal Miss Rate & Traffic Ratio:          0.01          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct Mapped
Cache Size    Lines      Bits       Miss    Traffic      Relative
(bytes)      Fetched   Fetched      Rate      Ratio       Traffic
 16384           27      3456     0.02%       0.09%        0.0009
  8192           27      3456     0.02%       0.09%        0.0009
  4096           27      3456     0.02%       0.09%        0.0009
  2048           27      3456     0.02%       0.09%        0.0009
  1024           27      3456     0.02%       0.09%        0.0009
   512           27      3456     0.02%       0.09%        0.0009

Tno  Uay Associative
Cache Size    Lines      Bits       Miss    Traffic      Relative
(bytes)      Fetched   Fetched      Rate      Ratio       Traffic
 16384           27      3456     0.02%       0.09%        0.0009
  8192           27      3456     0.02%       0.09%        0.0009
  4096           27      3456     0.02%       0.09%        0.0009
  2048           27      3456     0.02%       0.09%        0.0009
  1024           27      3456     0.02%       0.09%        0.0009
   512           27      3456     0.02%       0.09%        0.0009

Architecture:  IBM360-2  Reference  architecture:  self
```

Listing 3

```
*** Cache performance statistics ***   Benchmark: myfile
Instructions read from cache:  216962   (6658080 bits   832260 bytes)
Basic blocks traced: 14069, Relative No-Cache Traffic:        1.6421

LineSize===> 32 bits
Total lines in memory: 170,   Distinct lines referenced (approx) = 160
```

```
Estim. Optimal Miss Rate & Traffic Ratio:          0.07          0.08
Estim. Optimal Relative Traffic:        0.0013

Direct Mapped
Cache Size    Lines      Bits      Miss    Traffic     Relative
 (bytes)     Fetched   Fetched     Rate      Ratio      Traffic
   4096          171      5472     0.08%      0.08%       0.0013
   2048          171      5472     0.08%      0.08%       0.0013
   1024          171      5472     0.08%      0.08%       0.0013
    512          171      5472     0.08%      0.08%       0.0013
    256          171      5472     0.08%      0.08%       0.0013
    128       134883   4316256    62.17%     64.83%       1.0645

Two Uay Associative
Cache Size    Lines      Bits      Miss    Traffic     Relative
 (bytes)     Fetched   Fetched     Rate      Ratio      Traffic
   4096          171      5472     0.08%      0.08%       0.0013
   2048          171      5472     0.08%      0.08%       0.0013
   1024          171      5472     0.08%      0.08%       0.0013
    512          171      5472     0.08%      0.08%       0.0013
    256          171      5472     0.08%      0.08%       0.0013
    128       176165   5637280    81.20%     84.67%       1.3903

LineSize===> 64 bits
Total lines in memory: 85,   Distinct lines referenced (approx) = 80
Estim. Optimal Miss Rate & Traffic Ratio:          0.04          0.08
Estim. Optimal Relative Traffic:        0.0013

Direct Mapped
Cache Size    Lines      Bits      Miss    Traffic     Relative
 (bytes)     Fetched   Fetched     Rate      Ratio      Traffic
   4096           86      5504     0.04%      0.08%       0.0014
   2048           86      5504     0.04%      0.08%       0.0014
   1024           86      5504     0.04%      0.08%       0.0014
    512           86      5504     0.04%      0.08%       0.0014
    256           86      5504     0.04%      0.08%       0.0014
    128        71065   4548160    32.75%     68.31%       1.1217

Two Way Associative
Cache Size    Lines      Bits      Miss    Traffic     Relative
 (bytes)     Fetched   Fetched     Rate      Ratio      Traffic
   4096           86      5504     0.04%      0.08%       0.0014
   2048           86      5504     0.04%      0.08%       0.0014
   1024           86      5504     0.04%      0.08%       0.0014
    512           86      5504     0.04%      0.08%       0.0014
    256           86      5504     0.04%      0.08%       0.0014
    128        89895   5753280    41.43%     86.41%       1.4190

LineSize===> 128 bits
Total lines in memory: 42,   Distinct lines referenced (approx) = 40
Estim. Optimal Miss Rate & Traffic Ratio:          0.02          0.08
Estim. Optimal Relative Traffic:        0.0013

Direct Mapped
Cache Size    Lines      Bits      Miss    Traffic     Relative
 (bytes)     Fetched   Fetched     Rate      Ratio      Traffic
```

```
  4096           43      5504     0.02%       0.08%        0.0014
  2048           43      5504     0.02%       0.08%        0.0014
  1024           43      5504     0.02%       0.08%        0.0014
   512           43      5504     0.02%       0.08%        0.0014
   256           43      5504     0.02%       0.08%        0.0014
   128        41926   5366528    19.32%      80.60%        1.3236

Two Uay Associative
Cache Size    Lines      Bits      Miss     Traffic      Relative
(bytes)      Fetched   Fetched     Rate      Ratio        Traffic
  4096           43      5504     0.02%       0.08%        0.0014
  2048           43      5504     0.02%       0.08%        0.0014
  1024           43      5504     0.02%       0.08%        0.0014
   512           43      5504     0.02%       0.08%        0.0014
   256           43      5504     0.02%       0.08%        0.0014
   128        46957   6010496    21.64%      90.27%        1.4824

LineSize===> 256 bits
Total lines in memory: 21,   Distinct lines referenced (approx) = 20
Estim. Optimal Miss Rate & Traffic Ratio:          0.01          0.08
Estim. Optimal Relative Traffic:        0.0013

Direct Mapped
Cache Size    Lines      Bits      Miss     Traffic      Relative
(bytes)      Fetched   Fetched     Rate      Ratio        Traffic
  4096           22      5632     0.01%       0.08%        0.0014
  2048           22      5632     0.01%       0.08%        0.0014
  1024           22      5632     0.01%       0.08%        0.0014
   512           22      5632     0.01%       0.08%        0.0014
   256           22      5632     0.01%       0.08%        0.0014
   128        24527   6278912    11.30%      94.31%        1.5486

Two Way Associative
Cache Size    Lines      Bits      Miss     Traffic      Relative
(bytes)      Fetched   Fetched     Rate      Ratio        Traffic
  4096           22      5632     0.01%       0.08%        0.0014
  2048           22      5632     0.01%       0.08%        0.0014
  1024           22      5632     0.01%       0.08%        0.0014
   512           22      5632     0.01%       0.08%        0.0014
   256           22      5632     0.01%       0.08%        0.0014
   128        25360   6492160    11.69%      97.51%        1.6012

Architecture: fix32-0  Reference architecture: IBM360-2
```

Listing 4

```
*** Cache performance statistics ***   Benchmark: myfile
Instructions read from cache: 216962   (6658080 bits   832260 bytes)
Basic blocks traced: 14069, Relative Ho-Cache Traffic:        1.0000

LineSize===> 32 bits
Total lines in memory: 170,   Distinct lines referenced (approx) = 160
```

```
Estim. Optimal Miss Rate & Traffic Ratio:          0.07          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct  Mapped
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
  4096          171      5472     0.08%      0.08%       0.0008
  2048          171      5472     0.08%      0.08%       0.0008
  1024          171      5472     0.08%      0.08%       0.0008
   512          171      5472     0.08%      0.08%       0.0008
   256          171      5472     0.08%      0.08%       0.0008
   128       134883   4316256    62.17%     64.83%       0.6483

Tao  Uay Associative
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
  4096          171      5472     0.08%      0.08%       0.0008
  2048          171      5472     0.08%      0.08%       0.0008
  1024          171      5472     0.08%      0.08%       0.0008
   512          171      5472     0.08%      0.08%       0.0008
   256          171      5472     0.08%      0.08%       0.0008
   128       176165   5637280    81.20%     84.67%       0.8467

LineSize===>  64 bits
Total lines in memory: 85,   Distinct lines referenced (approx) = 80
Estim. Optimal Miss Rate & Traffic Ratio:          0.04          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct  Mapped
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
  4096           86      5504     0.04%      0.08%       0.0008
  2048           86      5504     0.04%      0.08%       0.0008
  1024           86      5504     0.04%      0.08%       0.0008
   512           86      5504     0.04%      0.08%       0.0008
   256           86      5504     0.04%      0.08%       0.0008
   128        71065   4548160    32.75%     68.31%       0.6831

Two  Uay Associative
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
  4096           86      5504     0.04%      0.08%       0.0008
  2048           86      5504     0.04%      0.08%       0.0008
  1024           86      5504     0.04%      0.08%       0.0008
   512           86      5504     0.04%      0.08%       0.0008
   256           86      5504     0.04%      0.08%       0.0008
   128        89895   5753280    41.43%     86.41%       0.8641

LineSize===> 128 bits
Total lines in memory: 42,   Distinct lines referenced (approx) = 40
Estim. Optimal Miss Rate & Traffic Ratio:          0.02          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct  Mapped
Cache Size    Lines      Bits      Miss    Traffic     Relative
(bytes)      Fetched   Fetched     Rate      Ratio      Traffic
```

```
  4096          43      5504     0.02%      0.08%        0.0008
  2048          43      5504     0.02%      0.08%        0.0008
  1024          43      5504     0.02%      0.08%        0.0008
   512          43      5504     0.02%      0.08%        0.0008
   256          43      5504     0.02%      0.08%        0.0008
   128       41926   5366528    19.32%     80.60%        0.8060

Two  Way Associative
Cache Size    Lines      Bits      Miss    Traffic      Relative
(bytes)      Fetched   Fetched      Rate      Ratio       Traffic
  4096          43      5504     0.02%      0.08%        0.0008
  2048          43      5504     0.02%      0.08%        0.0008
  1024          43      5504     0.02%      0.08%        0.0008
   512          43      5504     0.02%      0.08%        0.0008
   256          43      5504     0.02%      0.08%        0.0008
   128       46957   6010496    21.64%     90.27%        0.9027

LineSize===>  256 bits
Total lines in memory: 21,   Distinct lines referenced (approx) = 20
Estim. Optimal Miss Rate & Traffic Ratio:          0.01          0.08
Estim. Optimal Relative Traffic:        0.0008

Direct  Mapped
Cache Size    Lines      Bits      Miss    Traffic      Relative
(bytes)      Fetched   Fetched      Rate      Ratio       Traffic
  4096          22      5632     0.01%      0.08%        0.0008
  2048          22      5632     0.01%      0.08%        0.0008
  1024          22      5632     0.01%      0.08%        0.0008
   512          22      5632     0.01%      0.08%        0.0008
   256          22      5632     0.01%      0.08%        0.0008
   128       24527   6278912    11.30%     94.31%        0.9431

Two Uay Associative
Cache Size    Lines      Bits      Miss    Traffic      Relative
(bytes)      Fetched   Fetched      Rate      Ratio       Traffic
  4096          22      5632     0.01%      0.08%        0.0008
  2048          22      5632     0.01%      0.08%        0.0008
  1024          22      5632     0.01%      0.08%        0.0008
   512          22      5632     0.01%      0.08%        0.0008
   256          22      5632     0.01%      0.08%        0.0008
   128       25360   6492160    11.69%     97.51%        0.9751

Architecture:  fix32-0  Reference  architecture:  self
```

Listing 5

```
simulator:  srssim
version:    3.0
date:      16 May 88
time:       16:41:55
program:    myfile
```

```
simsim: heap: 0 static: 0 io: 0

status blocks calls rt recs io recs refs
static 15 1 70
completed   13968 1 24200 372 76512    --(1)

srssim:  stack[global[fmt,mmt],local] grs:2 lrs:6 ordered  othersr

traffic:   read nrite total
cpu to reg 0 0 0
mem to reg 0 0 0
cpu to global 37665 8595 46260

register set statistics:
size loads stores alloc calls a/c malloc cdepth m/c
6 0 0 4 1 4.00 4 1 4.00

register-set size distribution:
regs count relative
4 1 1.000 *******************************

call chain distribution
chain count relative
leaf 1 1.000 ********************************
1 0 0.000

--------------------------------------------------------------------------------

second level reference counts
traff: read write total read write total
access:  22800 7453 30253 1.000 1.000 1.000

access:  read write total read write total
global  22800 7452 30252 1.000 1.000 1.000 **********************
stack   0 1 1 0.000 0.000 0.000

--------------------------------------------------------------------------------

ecache:
transfer size: 4 bytes in 4 bytes out

acmask:  [access maintenance]
rdmask:  [static  global  stack   struct  heap    frame   ]
nrmask:  [static  global  stack   struct  heap    frame   ]

traff: read write total read write total
maint:  0 0 0 0.000 0.000 0.000
access:  22800 7455 30255 1.000 1.000 1.000
total:  22800 7455 30255 0.754 0.246 1          ---(2)

access:  read write total read write total
static  0 0 0 0.000 0.000 0.000
global  22800 7452 30252 1.000 1.000 1.000 **********************
stack   0 3 3 0.000 0.000 0.000
struct  0 0 0 0.000 0.000 0.000
heap    0 0 0 0.000 0.000 0.000
```

```
frame   0 0 0 0.000 0.000 0.000
io      0 0 0 0.000 0.000 0.000
m+a:  read write total read nrite total
static  0 0 0 0.000 0.000 0.000
global  22800 7452 30252 1.000 1.000 1.000 **********************
stack   0 3 3 0.000 0.000 0.000
struct  0 0 0 0.000 0.000 0.000
heap    0 0 0 0.000 0.000 0.000
frame   0 0 0 0.000 0.000 0.000
io      0 0 0 0.000 0.000 0.000

--------------------------------------------------------------------------------

memory
transfer size: 4 bytes
```

Listing 6

```
simulator:  mrssim
version:    2.0
date:      16  May 88
the:        16:44:45
program:    myfile

simsim: heap: 0 static: 0 io: 0

status blocks calls rt recs io recs refs
static 15 1 70
completed   13968 1 24200 372 76512   --(1)

--------------------------------------------------------------------------------

bufsim: segment[global[fmt,mmt],structure_stack,simple_stack]
globalbuffer[8]
mrs[2/6,4,rag,par_mem,load[static],flush[static],]

traffic:   read write total
cpu to global 37665 8595 46260

windows words ovf unf ov perc un perc
1 6 0 0 0.000 0.000
2 12 0 0 0.000 0.000
4 24 0 0 0.000 0.000
8 48 0 0 0.000 0.000
16 96 0 0 0.000 0.000
32 192 0 0 0.000 0.000
64 384 0 0 0.000 0.000
128 768 0 0 0.000 0.000

misses type read write total rrd rwr rtot
1 local   0 0 0 0 0 0
1 global  0 0 0 0 0 0
```

```
1 nlocal 0 0 0 0 0 0
1 link    0 0 0 0 0 0
1 call    0 0 0 0 0 0
1 return 0 0 0 0 0 0
1 maint   0 0 0 0 0 0
24 total 0 0 0 0 0
2 local   0 0 0 0 0 0
2 global 0 0 0 0 0 0
2 nlocal 0 0 0 0 0 0
2 link    0 0 0 0 0 0
2 call    0 0 0 0 0 0
2 return 0 0 0 0 0 0
2 maint   0 0 0 0 0 0
48 total 0 0 0 0 0
4 local   0 0 0 0 0 0
4 global 0 0 0 0 0 0
4 nlocal 0 0 0 0 0 0
4 link    0 0 0 0 0 0
4 call    0 0 0 0 0 0
4 return 0 0 0 0 0 0
4 maint   0 0 0 0 0 0
96 total 0 0 0 0 0
8 local   0 0 0 0 0 0
8 global 0 0 0 0 0 0
8 nlocal 0 0 0 0 0 0
8 link    0 0 0 0 0 0
8 call    0 0 0 0 0 0
8 return 0 0 0 0 0 0
8 maint   0 0 0 0 0 0
192 total 0 0 0 0 0
16 local   0 0 0 0 0 0
16 global 0 0 0 0 0 0
16 nlocal 0 0 0 0 0 0
16 link    0 0 0 0 0 0
16 call    0 0 0 0 0 0
16 return 0 0 0 0 0 0
16 maint   0 0 0 0 0 0
384 total 0 0 0 0 0
32 local   0 0 0 0 0 0
32 global 0 0 0 0 0 0
32 nlocal 0 0 0 0 0 0
32 link    0 0 0 0 0 0
32 call    0 0 0 0 0 0
32 return 0 0 0 0 0 0
32 maint   0 0 0 0 0 0
768 total 0 0 0 0 0
64 local   0 0 0 0 0 0
64 global 0 0 0 0 0 0
64 nlocal 0 0 0 0 0 0
64 link    0 0 0 0 0 0
64 call    0 0 0 0 0 0
64 return 0 0 0 0 0 0
64 maint   0 0 0 0 0 0
1536 total 0 0 0 0 0
128 local   0 0 0 0 0 0
128 global 0 0 0 0 0 0
```

```
128 nlocal  0 0 0 0 0 0
128 link    0 0 0 0 0 0
128 call    0 0 0 0 0 0
128 return  0 0 0 0 0 0
128 maint   0 0 0 0 0 0
3072 total 0 0 0 0 0

hits type top rd top wr top rw ali rd ali wr ali rw tot rw
1 local   0 0 0 0 0 0 0
1 global 0 0 0 0 0 0 0
1 nlocal 0 0 0 0 0 0 0
1 link    0 0 0 0 0 0 0
1 call    0 2 2 0 0 0 2
1 return 0 0 0 0 0 0 0
24 total 0 2 2 0 0 0 2
2 local   0 0 0 0 0 0 0
2 global 0 0 0 0 0 0 0
2  nlocal  0  00  00  00
2 link    0 0 0 0 0 0 0
2 call    0 2 2 0 0 0 2
2  return  0  0000  00
48 total 0 2 2 0 0 0 2
4 local   0 0 0 0 0 0 0
4 global 0 0 0 0 0 0 0
4 nlocal  0  0000  00
4 link    0 0 0 0 0 0 0
4 call    0 2 2 0 0 0 2
4 return 0 0 0 0 0 0 0
96 total 0 2 2 0 0 0 2
8 local   0 0 0 0 0 0 0
8 global 0 0 0 0 0 0 0
8nlocal  0  0000  00
8 link    0 0 0 0 0 0 0
8 call    0 2 2 0 0 0 2
8 return 0 0 0 0 0 0 0
192 total 0 2 2 0 0 0 2
16 local   0 0 0 0 0 0 0
16 global 0 0 0 0 0 0 0
16 nlocal 0 0 0 0 0 0 0
16 link    0 0 0 0 0 0 0
16 call    0 2 2 0 0 0 2
16 return 0 0 0 0 0 0 0
384 total 0 2 2 0 0 0 2
32 local   0 0 0 0 0 0 0
32 global 0 0 0 0 0 0 0
32 nlocal 0 0 0 0 0 0 0
32 link    0 0 0 0 0 0 0
32 call    0 2 2 0 0 0 2
32 return 0 0 0 0 0 0 0
768 total 0 2 2 0 0 0 2
64 local   0 0 0 0 0 0 0
64 global. 0 0 0 0 0 0 0
64 nlocal 0 0 0 0 0 0 0
64 link    0 0 0 0 0 0 0
64 call    0 2 2 0 0 0 2
64 return 0 0 0 0 0 0 0
```

```
1536 total 0 2 2 0 0 0 2
128 local   0 0 0 0 0 0 0
128 global  0 0 0 0 0 0 0
128 nlocal  0 0 0 0 0 0 0
128 link    0 0 0 0 0 0 0
128 call    0 2 2 0 0 0 2
128 return  0 0 0 0 0 0 0
3072 total 0 2 2 0 0 0 2

-------------------------------------------------------------------------------

second level reference counts
traff: read write total read nrite total
access:  22800 7452 30252 1.000 1.000 1.000

access:  read write total read write total
global  22800 7452 30252 1.000 1.000 1.000 **********************

-------------------------------------------------------------------------------

ecache:
transfer size: 4 bytes in 4 bytes out

acmask :  [access maintenance]
rdmask:  [static  global  stack   struct  heap    frame   ]
wrmask:  [static  global  stack   struct  heap    frame   ]

traff: read nrite total read write total
maint:  0 0 0 0.000 0.000 0.000
access:  22800 7452 30252 1.000 1.000 1.000
total:  22800 7452 30252 0.754 0.246 1            --(2)

access :  read nrite total read write total
static  0 0 0 0.000 0.000 0.000
global  22800 7452 30252 1.000 1.000 1.000 **********************
stack   0 0 0 0.000 0.000 0.000
struct  0 0 0 0.000 0.000 0.000
heap    0 0 0 0.000 0.000 0.000
frame   0 0 0 0.000 0.000 0.000
io      0 0 0 0.000 0.000 0.000
m+a:  read write total read write total
static  0 0 0 0.000 0.000 0.000
global  22800 7452 30252 1.000 1.000 1.000 **********************
stack   0 0 0 0.000 0.000 0.000
struct  0 0 0 0.000 0.000 0.000
heap    0 0 0 0.000 0.000 0.000
frame   0 0 0 0.000 0.000 0.000
io      0 0 0 0.000 0.000 0.000

-------------------------------------------------------------------------------
memory
transfer size: 4 bytes
```

Listing 7

```
simulating: myfile
dynamic: 30255 (refs)

cache: bl size bl mar writes alloc dp in dp out subblock
asscch 32 1024 count rd/wr 4 4 0

rdmask  [static  global  stack   struct  heap    frame   ]
nrmask  [static  global  stack   struct  heap    frame   ]

traff: read write total read write total
static  0 0 0 0.000 0.000 0.000
global  22800 7452 30252 1.000 1.000 1.000 *********************
stack   0 3 3 0.000 0.000 0.000
struct  0 0 0 0.000 0.000 0.000
heap    0 0 0 0.000 0.000 0.000
frame   0 0 0 0.000 0.000 0.000
io      0 0 0 0.000 0.000 0.000
imm     0 0 0 0.000 0.000 0.000
total 22800 7455 30255 0.754 0.246 1.000 *********************

size segment rd miss wr miss tt miss r ratio w ratio t ratio
1*32 global 11430 1080 12510 0.501 0.145 0.414 ******
stack   0 2 2 0.000 0.667 0.667 *********
32 total11430 1082 12512 0.501 0.145 0.414 ******

2*32 global 2403 101 2504 0.105 0.014 0.083 *
stack   0 2 2 0.000 0.667 0.667 *********
64  total 2403 103 2506 0.105 0.014 0.083 *

4*32 global 717 101 818 0.031 0.014 0.027
stack   0 2 2 0.000 0.667 0.667 *********
128 total 717 103 820 0.031 0.014 0.027

8*32 global 540 19 559 0.024 0.003 0.018
stack   0 2 2 0.000 0.667 0.667 *********
256  total 540 21 561 0.024 0.003 0.019

16*32 global 0 14 14 0.000 0.002 0.000
stack   0 2 2 0.000 0.667 0.667 *********
512  total 0 16 16 0.000 0.002 0.001

32*32 global  0 14 14 0.000 0.002 0.000
stack  0  2  2 0.000 0.667 0.667 *********
1024  total 0 16 16 0.000 0.002 0.001

64*32 global  0 14 14 0.000 0.002 0.000
stack  0  2  2 0.000 0.667 0.667 *********
2048  total 0 16 16 0.000 0.002 0.001

128*32 global  0 14 14 0.000 0.002 0.000
stack  0  2  2 0.000 0.667 0.667 *********
4096 total 0 16 16 0.000 0.002 0.001
```

```
256*32 global  0 14 14 0.000 0.002 0.000
stack   0 2 2 0.000 0.667 0.667 *********
8192 total  0 16 16 0.000 0.002 0.001

512*32 global  0 14 14 0.000 0.002 0.000
stack   0 2 2 0.000 0.667 0.667 *********
16384 total 0 16 16 0.000 0.002 0.001

1024432 global  0  14  14  0.000  0.002  0.000
stack   0 2 2 0.000 0.667 0.667 *********
32768 total  0 16 16 0.000 0.002 0.001

size dirty cp rat cp rd cp wr cp tot wt rd wt wr wt tot
1  4361 0.144 100096 34888 134984 100096 7455 107551
2 1684 0.056 20048 13472 33520 20048 7455 27503
4 731 0.024 6560 5848 12408 6560 7455 14015   --(1)
8 504 0.017 4488 4032 8520 4488 7455 11943
16 16 0.001  128  128  256  128  7455  7583
32  16  0.001 128 128 256 128 7455 7583
64 16 0.001  128  128  256  128  7455  7583
128  16  0.001 128 128 256 128 7455 7583
256  16  0.001 128 128 256 128 7455 7583
512 16 0.001  128  128  256  128  7455  7583
1024  16  0.001 128 128 256 128 7455 7583
```

Listing 8

```
simulating: myfile
dynamic:  30255 (refs)

cache:  bl size bl max writes alloc repl assoc dp in dp out subsize
asssub 32 1024 count rd/wr fifo 4 4 4 4

rdmask:  [static   global  stack   struct  heap    frame   ]
nrmask:  [static   global  stack   struct  heap    frame   ]

traff: read write total read write total
static  0 0 0 0.000 0.000 0.000
global  22800 7452 30252 1.000 1.000 1.000 **********************
stack   0 3 3 0.000 0.000 0.000
struct  0 0 0 0.000 0.000 0.000
heap    0 0 0 0.000 0.000 0.000
frame   0 0 0 0.000 0.000 0.000
io      0 0 0 0.000 0.000 0.000
imm     0 0 0 0.000 0.000 0.000
total 22800 7455 30255 0.754 0.246 1.000 **********************

size segment rd miss wr miss tt miss r ratio w ratio t ratio
4-way global  5235  120 5355 0.230 0.016 0.177 **
1*32   stack  0   3   3  0.000 1.000 1.000 **************
128 total 5235 123 5358  0.230  0.016  0.177  ** --(1)
```

```
4-way global  3761  102 3863 0.165 0.014 0.128 **
2*32  stack  0  3  3 0.000 1.000 1.000  **************
256 total 3761 105 3866 0.165 0.014 0.128 **

4-way global  0 102 102 0.000 0.014 0.003
4432 stack   0 3 3 0.000 1.000 1.000 **************
512 total 0 105 105 0.000 0.014 0.003

4-way global  0 102 102 0.000 0.014 0.003
8*32 stack   0 3 3 0.000 1.000 1.000 **************
1024 total 0 105 105 0.000 0.014 0.003

4-way global  0 102 102 0.000 0.014 0.003
16432  stack   0 3 3 0.000 1.000 1.000 **************
2048 total 0 105 105 0.000 0.014 0.003

4-way global  0 102 102 0.000 0.014 0.003
32*32 stack   0 3 3 0.000 1.000 1.000 **************
4096 total 0 105 105 0.000 0.014 0.003

4-way global  0 102 102 0.000 0.014 0.003
64*32 stack   0 3 3 0.000 1.000 1.000 **************
8192 total 0 105 105 0.000 0.014 0.003

4-way global  0 102 102 0.000 0.014 0.003
128*32 stack   0 3 3 0.000 1.000 1.000 **************
16384 total 0 105 105 0.000 0.014 0.003

4-way global  0 102 102 0.000 0.014 0.003
256*32  stack   0 3 3 0.000 1.000 1.000 **************
32768 total 0 105 105 0.000 0.014 0.003

4-way global  0 102 102 0.000 0.014 0.003
512*32 stack   0 3 3 0.000 1.000 1.000 **************
65536 total 0 105 105 0.000 0.014 0.003

4-way  global 0 102 102 0.000 0.014 0.003
1024432 stack   0 3 3 0.000 1.000 1.000 **************
131072 total 0 105 105 0.000 0.014 0.003

a size dirty cp rat cp rd cp ur cp tot ut rd ut wr wt tot
4 1 3921 0.130 5358 3921 9279 5358 7455 12813   --(2)
4 2 3031 0.100 3866 3031 6897 3866 7455 11321
4 4 105 0.003 105 105 210 105 7455 7560
4 8 105 0.003 105 105 210 105 7455 7560
4 16 105 0.003 105 105 210 105 7455 7560
4 32 105 0.003 105 105 210 105 7455 7560
4 64 105 0.003 105 105 210 105 7455 7560
4 128 105 0.003 105 105 210 105 7455 7560
4 256 105 0.003 105 105 210 105 7455 7560
4 512 105 0.003 105 105 210 105 7455 7560
4 1024 105 0.003 105 105 210 105 7455 7560
```

Listing 9