PDATS
Lossless Address Trace Compression For Reducing File Size And Access Time

Eric E. Johnson and Jiheng Ha
Parallel Architecture Research Lab
New Mexico State University
ejohnson@nmsu.edu

ABSTRACT

The tremendous storage space required for a useful data base of traces has driven a search for trace compaction techniques. In this paper we present an information-lossless trace compression scheme that can reduce both storage space and access time by an order of magnitude or more, compared to ASCII-format traces, without discarding either references or inter-reference timing from the original trace.

This technique has been selected as the standard trace format for an extensive new trace data base that will be made accessible to the international research and teaching community.

INTRODUCTION

Trace driven simulation is often a cost-effective way to estimate the performance of computer system designs. Especially when designing a cache, TLB, or paging system, trace driven simulation is a popular way to obtain an acceptable estimation of performance before a system is built. However, to obtain useful results from a trace driven simulation of a two-level cache (or other similarly complicated system), traces with millions of references are needed. With on the order of ten bytes often required to store each reference, trace file sizes of tens to hundreds of megabytes are not unusual. Further, because general-purpose computer systems must execute a variety of programs well, trace-driven simulation of these systems requires the use of representative traces from each application area of interest, with a different set of traces needed to exercise each subsystem to be evaluated. Thus, it is quite easy for a small library of traces to consume gigabytes of secondary storage unless some means is employed to compress the traces.

Another consideration in trace-driven simulation is the time required to read the references from the trace file. Reducing the trace file size results in less data movement from secondary storage to main memory, and therefore less time required for file I/O in a trace-driven simulation. However, this time savings comes at a cost of CPU time to decompress the trace, so the net effect on simulation time depends upon the complexity of the decompression algorithm.

In the remainder of the Introduction we discuss approaches to compressing trace files reported in the literature. The following sections present the PDATS trace compression scheme and an evaluation of its performance.

This work was supported in part by the U.S. Army Research Office under grant DAAH04–93-0229.

Generic File Compression

The Unix™ `compress` utility uses the well-known Lempel-Ziv (LZ) algorithm to compress files, and has been widely used to reduce the size of files to be copied over low-bandwidth networks. For example, the trace files available to support the Hennessy and Patterson computer architecture text [4] are available for anonymous ftp in compressed form. In this case, LZ compression achieved a (roughly) 5:1 compression ratio for a five-fold savings in transfer time and storage space.

However, such compressed traces must be uncompressed, either before or during each simulation. If the files are uncompressed before use, they consume roughly five times as much disk space, but simulations run faster than if the traces are left compressed and piped through `zcat` (for uncompression) into the simulator.

Trace Filtering

An approach that reduces both trace size and access time is deleting trace references that carry little information. However, throwing away trace references reduces accuracy of trace simulations. Smith proposed two methods to reduce trace size [9]:

a) *Stack deletion* which throws away references that are hits to the top D levels of an LRU stack. The resulting information loss is reported to introduce an error rate in paging simulations on the order of 1% or less while producing a compression ratio in the range of 1.05 to 4.

b) *Snapshots* which only record a set of references every T references. The range of error rates caused by the snapshot method is similar to the first method while producing a compression ratio of 5 to 100. These two schemes have been shown to be primarily suitable for paging studies [1].

Puzak described a trace stripping scheme that employs a small direct-mapped filter cache to remove redundant references. This technique introduces no error in simulations of caches with the same line size as the filter cache [6]. However, when the line size is varied, significant errors in miss ratios are produced [3].

Agarwal introduced a two step trace compaction scheme: a cache filter followed by a block filter [1]. Errors in simulated miss ratios ranged from 1 to -16%, while the compression ratio reached 15 under some configurations of the cache filter and block filter.

In addition to the errors produced in cache simulations, another problem with these trace filtering techniques is that traces are used to drive simulations of not only memory hierarchies, but also of many other subsystems of modern computers. For

example, Malik [5] describes the use of traces of instruction streams to evaluate novel ALU architectures. Trace filtering can, of course, be applied to discard low-information segments of traces for each subsystem to be evaluated, but the need to store multiple filtered traces produced from each original trace largely eliminates the storage space reduction that accrues from filtering.

The fundamental problem in trace filtering is that information is lost in the process; we would prefer a trace *compression* technique that recodes the references in an information-lossless manner. Such compressed traces would retain their general applicability, with no induced errors.

Lossless Trace Compression

Samples proposed the Mache scheme [7] to derive no-loss trace compaction by converting memory references into an address difference trace, with an average compression ratio for data/ instruction traces of 2.5. After that, LZ compression [10] was used to compress the trace file size by another factor of 4. The Mache scheme focused on reducing trace size without losing any information, but did not address reduction of simulation time. However, the time to read a trace into a simulator can be of equal concern when very long traces are used.

PDATS TRACE COMPRESSION

Technique

The PDATS (Packed Differential Address and Time Stamp) scheme was developed at the Parallel Architecture Research Lab (PARL) of NMSU, with a goal of reducing both trace size and simulation time without distorting the reference stream contained in the original trace. This can be viewed as a coding problem, in which bandwidth reduction is to be balanced against decoding complexity.

Although PDATS was developed independently of Mache, both use a similar idea for data compression: rather than storing an address sequence explicitly, only the address differences between successive references of the same type (e.g., fetch, read, or write) are stored. Due to the locality of reference exhibited by most programs, these differences are significantly smaller on average than the absolute addresses they replace.

Unlike Mache, PDATS uses variable length records and offset run length coding, resulting in greater compression than the basic Mache scheme. This, in turn, eliminates the need for time-consuming LZ decompression for traces in active use, although LZ compression is still useful for further reducing the sizes of infrequently-used traces.

Table 1 illustrates the steps used to produce a PDATS trace from a dinero trace. (The dinero file format is used by Mark Hill's popular `dineroIII` cache simulator.)

The first column in Table 1 is the type of reference: 0 represents a data read, 1 a data write, and 2 an instruction fetch. This reference type field is also used in the PDATS file. The memory locations accessed are listed in the second column. PDATS converts these absolute addresses to address offsets, which are the differences between successive references *of the same type*, represented in two's complement using the minimum number of bytes required to hold the offset. The first address of each type is simply reproduced in the output file. When the original trace contains time stamps, non-negative offset encoding is employed to compress these.

Note that when successive references are of the same type and maintain a constant offset, a single instance of the type and offset specification, together with the run length, suffices to describe the entire sequence of references.

Table 1: Example of Trace Processing

Input Trace		PDATS Trace		
Type	Address	Type	Offset	Run Length
2	430d70	2	430d70	1
2	430d74	2	4	1
2	415130	2	-1bc44	1
0	1000acac	0	1000acac	1
2	415134	2	4	4
2	415138			
2	41513c			
2	415140			
2	430c20	2	1bae0	1
2	430c24	2	4	1
1	7fff00ac	1	7fff00ac	1
2	430c28	2	4	1
1	7fff00a8	1	-4	1
2	430c2c	2	4	1
1	7fff00a4	1	-4	1
2	430c30	2	4	1
1	7fff009c	1	-8	1
2	430c34	2	4	1
1	7fff00a0	1	4	1
2	430c38	2	4	1
1	7fff0098	1	-4	1
2	430c3c	2	4	8
2	430c40			
2	430c44			
2	430c48			
2	430c4c			
2	430c50			
2	430c54			
2	430c58			
1	7fff00b4	1	1c	1

File Format

A PDATS file is in a binary format consisting of a file header followed by variable-length records. The header of a PDATS trace file begins with a 16-bit magic number that identifies the specific PDATS file format. Two magic numbers are currently used for uniprocessor traces: `0xff01` indicates a file without time stamps, and `0xff03` indicates a file with time stamps. The remainder of the header is optional, and may contain records describing the source of the trace or other useful information.

The rest of a PDATS file consists of variable length records (from 1 to 8 bytes) in the following format:

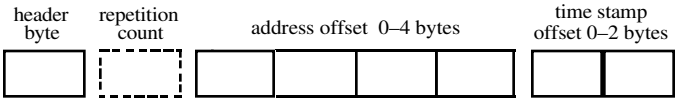

Figure 1: PDATS Record Structure

The structure of the header byte is shown in Figure 2, with the field encodings listed in Table 2.

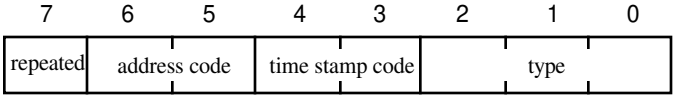

Figure 2: PDATS Header Byte Structure

The most straightforward approach to encoding address and time stamp offsets would simply use the minimum number of bytes necessary to hold each offset, with a tag in the header byte to indicate the number of bytes used. The coding efficiency of PDATS has been increased over this simple scheme by reserving a few codes in the header byte for specific high-probability offsets:

c) Approximately 85% of instruction fetches (from 32-bit microprocessors, both CISC and RISC) reference sequential memory words, as do from 9% to 38% of data loads and stores, making 4 byte offsets extremely common.

d) In RISC microprocessors, the time between instruction references is nearly always 1 clock cycle, making a time stamp offset of 1 very likely in RISC traces.

e) In Harvard-architecture microprocessors, data references can occur simultaneously with instruction references, leading to frequent occurrences of time stamp offsets of 0.

When these cases occur, the corresponding offset is encoded entirely in the header byte, with the result that many references can be encoded in a 1-byte record.

When the repeat flag is set, the byte following the header byte specifies the number of times that this record is repeated (contiguously) in the original trace. This run-length coding can produce significant compression of instruction sequences with minimal computation required to regenerate the original reference stream.

Table 2: Header Byte Encoding

type	(bit 2 - 0)
0	user data read
1	user data write
2	user instruction fetch
3	escape record (unknown access type)
4	supervisor data read
5	supervisor data write
6	supervisor instruction fetch
7	interrupt acknowledge/co-processor
time stamp code	**(bit 4 - 3)**
00	time stamp increases by 0
01	time stamp increases by 1
10	time stamp offset is 2 - 255 (occupies 1 byte)
11	time stamp offset is 256 - 65536 (2 bytes)
address code	**(bit 6 - 5)**
00	address offset is exactly 4
01	address offset encoded in 1 byte (-128 to +127)
10	address offset encoded in 2 bytes
11	address offset encoded in 4 bytes
repeat	**(bit 7)**
0	no repetition (repeat count byte absent)
1	repeat these offsets (repeat count present)

EVALUATION

We have evaluated the effectiveness of the PDATS technique in compressing several other trace formats by comparing the file sizes and the time needed to read references from the original and compressed traces. In this section, we first describe the other trace formats and discuss the traces and methods used in the evaluation, followed by the compression and speedup results.

Traces Used in Evaluation

Table 3 summarizes the formats of the traces used in our evaluation. In addition to the popular dinero format, we evaluated two in-house derivatives of dinero and the native (binary) file format produced by our hardware trace collection system. Dinero and its derivatives store trace references in variable-length ASCII character strings, with space characters embedded to separate fields and a newline character used to separate records. Our derivatives extend the basic dinero record format to include a time stamp (Green Stamp) or data needed to reconstruct a trace that has been filtered (Schieber).

Table 3: Trace Formats Used in Evaluation

Trace Format	Source	Code	Record Size (bytes)	Time Stamps
DAS	Tektronix Data Acquisition System	binary	20	yes
dinero	U.C. Berkeley (M. Hill)	ASCII	3–10	no
Schieber	NMSU (C. Schieber) used for filtered traces	ASCII	9–16	no
Green Stamp	NMSU (M. Lumeyer)	ASCII	6–20	yes

The trace files used in our evaluation of PDATS are described in Table 4. For each trace format, we selected traces that represent a variety of program reference behaviors. When possible, we included a trace of a C compiler (relatively irregular behavior), a Spice simulation (a mixture of looping and branching), and a third program that exhibits fairly repetitive behavior. We include traces of both RISC and CISC processors, filtered and unfiltered traces, and traces that include supervisor and interrupt activity as well as user-only traces.

(Many of these traces are available in the NMSU TraceBase. Send e-mail to tracebase@nmsu.edu for more information.)

Table 4: Traces Used in Evaluation

Format/CPU	Trace	References	Program	User only	Filtered
DAS	gcc_all	131,009	gcc		
68020	spice_all	98,301	spice	n	n
(CISC)	gpssh_all	98,301	gpssh		
Schieber	gccbeg1	258,122	gcc		
68020	spmid1	258,252	spice	n	y
(CISC)	espbeg1	260,794	espresso		
Green Stamp 29050 (RISC)	espm2	448,730	espresso	y	n
dinero	cc1	1,000,002	gcc		
DLX	spice	1,000,001	spice	y	n
(RISC)	tex	832,477	LaTex		
dinero	spic.000	446,701	spice	n	n
VAX (CISC)	mul8.003	429,432	mul8		

These relatively small traces (no more than 1 million references each) were selected for efficiency in the initial evaluation of the PDATS technique. Because PDATS operates on very small neighborhoods of the trace file, the results obtained from these traces also apply to larger traces, as discussed at the end of this section.

To individually evaluate the various components of the PDATS technique, each trace was recoded into four different formats:

a) *Binary*: Each trace was first converted into an uncompressed binary form. Each reference was stored as 1 byte of type, 4 bytes of address, and 4 bytes of time stamp (if time stamps were present in the original file). Reduction of all of the input traces to this single uncompressed format permits comparison of the information density in the original formats.

b) *No Repeats*: The traces were compressed into packed differential form (Figure 1) without run-length coding. The ratio of the resulting file size to the uncompressed binary file size yields a measure of the spatial locality in the trace.

c) *PDATS*: The traces were compressed into packed differential form, with run-length coding used whenever possible. This is the normal mode for PDATS. Any improvement in compression over the version without repeat records is a result of sequentiality in the traces.

d) *PDATS.Z*: After compressing the trace using PDATS, LZ compression was applied to further compress the trace file.

The remainder of this section reports our findings in how effectively PDATS reduces the storage space and access time required for traces.

Evaluation of Compression

The results of compressing the twelve example traces are presented below, grouped by original trace format and CPU type. For each group, the sizes of the original and transformed files are shown graphically; numerical results for all groups are listed in Table 5.

Table 5: PDATS Compression Results

Trace	O'head	Spatial Local.	Seq.	PDATS Compr.	PDATS + LZ
	binary / original	binary / no rep	no rep / PDATS	original / PDATS	original / PDATS.Z
DAS	0.55	2.56	1.05	6.00	16.95
Schieber	0.67	1.61	1.00	4.91	8.48
Green Stamp	0.43	4.13	1.75	12.66	34.48
dinero (DLX)	0.47	2.19	1.33	5.51	24.59
dinero (VAX)	0.52	2.01	1.15	4.83	12.46

Compression of DAS Format Traces

The DAS format traces, `gcc_all`, `spice_all`, and `gpssh_all`, were captured in real-time from executions of the corresponding programs on an instrumented Sun–3/60 (68020 CISC CPU) running SunOS 4.0.3. These traces include user,

supervisor, and interrupt activity. Every reference carries a time stamp. The DAS results are shown in Figure 3.

The original files are 2.23 times the size of the uncompressed binary files, for an "overhead" of 55%. The binary files, in turn, are 2.56 times the size of the files without repeat records, and the use of repeat records reduces the file sizes by another 5%. The average compression ratio using PDATS alone is 6.00; with the addition of LZ compression, the ratio climbs to 16.95.

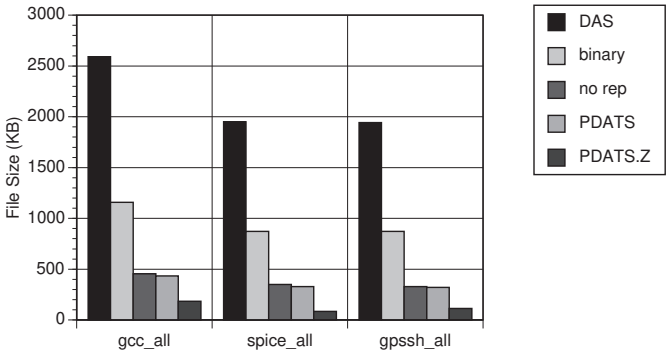

Figure 3: Compression of DAS Format Traces

Compression of Schieber Format Traces

The Schieber format traces, `gccbeg1`, `spmid1`, and `espbeg1` were captured and filtered in real time from the instrumented Sun–3/60 described above, using our RATCHET system [8]. Each trace is the contents of one of four hardware buffers, converted into a format derived from dinero that includes reference type, address, transfer size, and a flag that pulses every 2^{16} actual (not filtered) references. Time stamps are not included.

The results of compressing these traces are shown in Figure 4. As expected, the sequentiality of these traces has been completely filtered out, along with much of the spatial locality.

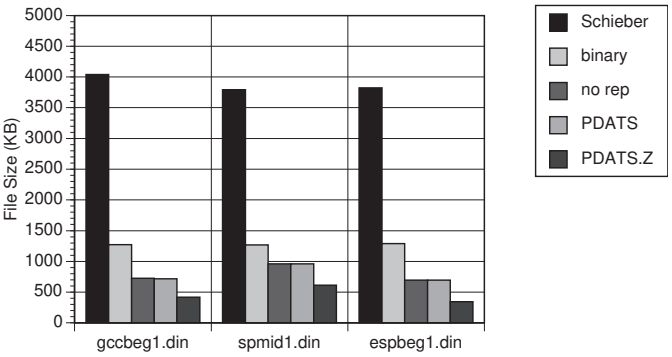

Figure 4: Compression of Schieber Format Traces

Compression of Green Stamp Format Traces

The Green Stamp trace `espm2` was generated from the SPEC89 program `espresso`, running on an Am29050 processor simulator. Supervisor and interrupt references are not included in the trace. Time stamps are included.

The Green Stamp results are shown in Figure 5. This RISC processor is capable of extended instruction bursts and simultaneous instruction and data accesses (Harvard architecture), resulting in strong sequentiality (75%) and frequent time stamp differences of exactly 1. As a result, PDATS was quite effective in compressing this trace, achieving a compression ratio of 12.66 alone, and 34.48 when teamed with LZ.

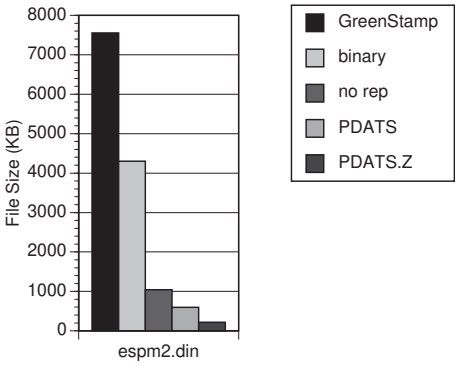

Figure 5: Compression of Green Stamp Format Traces

Compression of Dinero Format Traces

DLX is a "paper machine" used in a popular computer architecture text [4]. This processor incorporates characteristics of many of the first generation RISC processors. The `cc1.din`, `spice.din`, and `tex.din` traces were produced by a DLX simulator, and are available to users of this text. These are in dinero format, and include only reference type and address fields in each record.

The results of compressing these traces using PDATS are shown in Figure 6. The overhead in the original trace format (47%) and the sequentiality (33%) are similar to the other RISC trace, `espm2` in Green Stamp format. However, the "spatial locality" measure (2.9) is substantially lower in the dinero traces due to the absence of time stamps.

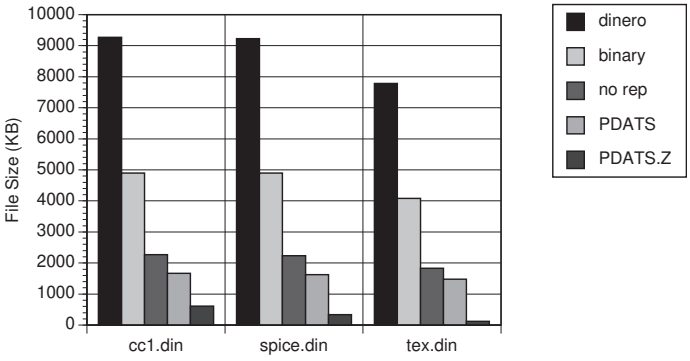

Figure 6: Compression of Dinero Traces from DLX

The `spic.000.din` and `mul8.003.din` traces are also in dinero format, and were obtained from a VAX using the ATUM technique [2]. Supervisor references are included, but time stamps are not. Results of compressing these traces are shown in Figure 7. The VAX results are similar to the other CISC (68020) results seen in the DAS and Schieber format traces.

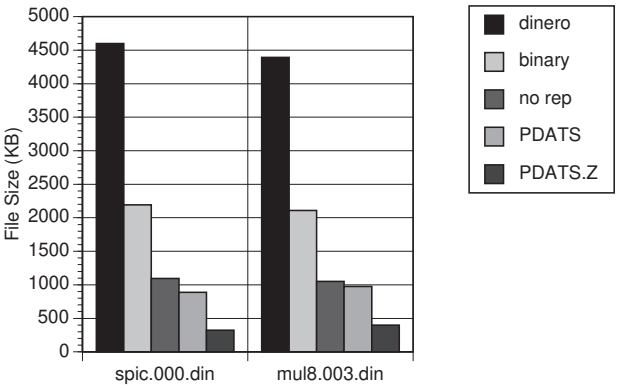

Figure 7: Compression of Dinero Traces from VAX

Compression of Longer Traces

Because PDATS operates on the scale of basic blocks within a trace, one would expect that the compression results obtained for the "toy" traces above will also apply to arbitrarily long "production" traces. As a check on this intuition, we compressed several traces of useful size, with the results shown in Table 6. (These traces were collected by Nadeem Malik of IBM from SPEC92 programs and the Unix utility `awk` running on an R2000 (RISC) processor. The `cexp` trace comes from a C compiler, while the `mdljd` program is a double-precision FORTRAN code. We converted them to the various formats shown.)

When these results are compared with the compression ratios obtained for the shorter RISC traces, we see that the compression ratios improved somewhat; this may be due to the greater repetitiveness likely in longer traces. We believe that the compression ratios obtained for most traces of interest will fall within the range of the results in Tables 5 and 6.

Table 6: Compression Results for Longer Traces

Trace	Dinero trace size	References	PDATS Compr.	PDATS + LZ
cexp	166,702,246	18,782,177	6.45	20.3
mdljd	738,116,943	84,233,871	8.49	61.2
awk2	757,070,302	86,435,124	6.40	32.7

Summary of Compression Results

From the results of this study we conclude that for traces of RISC processors, use of the PDATS technique should result in compression ratios in the range of 5 – 12, while CISC traces should be compressed by a factor of 5 – 6. Compared to Mache alone (i.e., without LZ), PDATS achieves 2 to 3 times more compression. When LZ compression is applied after PDATS, we expect a 3- to 5-fold further reduction in file size.

In general, we find that the following characteristics of a trace will affect its compression under PDATS:

a) Traces from RISC processors have higher compression ratios than those from CISC processors. This is probably due to the greater proportion of instruction references in a RISC trace, as well as the longer basic blocks, both of which increase sequentiality.

b) Unfiltered traces have better compression ratios than filtered traces, because most of the spatial locality and sequentiality has been removed from filtered traces.

c) User-only traces have better compression ratios than traces that include supervisor or interrupt references as well (complete traces). This may be due to irregular access patterns in supervisor and interrupt code.

d) Traces with time stamps have better compression ratios than those without time stamps, because time stamps usually increase by small amounts. Furthermore, when time stamp offsets are 0 or 1, PDATS achieves excellent compression by encoding the time stamp offset entirely within the header byte of the record.

Evaluation of Speedup

The time to read traces compressed using PDATS was evaluated two ways. First, we measured the time to simply read traces in original, binary, and PDATS formats, including the conversion of each reference into its integer components. These measurements clearly show the speedup resulting from PDATS compression; however, trace consuming programs normally process each reference after it is read, so these speedups will only be approximated in practice. In the second phase of the time evaluation, we modified the dinero III cache simulator to read PDATS traces, and compared the running time of cache simulations using equivalent traces in dinero and PDATS formats.

The access times were measured by placing each trace on a SCSI disk attached to a Sun 4/280 file server running SunOS 4.1.1, and measuring the user plus system time spent in reading and converting that trace. These times were collected while no other users were logged in to the machine, although background processes were executing normally. The results are summarized in Table 7, where the times for individual traces are summed within the groups.

Table 7: Summary of Access Time Results

Trace Format	Time (original)	Time (PDATS)	PDATS speedup
DAS	10.78 s	7.88 s	1.37
Schieber	126	14.0	9.01
Green Stamp	157	5.7	27.5
dinero (DLX)	382	35.3	10.8
dinero (VAX)	128	13.8	9.28

Access Time for DAS Format Traces

For DAS traces, the speedup achieved by PDATS over the original uncompressed file or the binary file is small, despite the significant compression achieved (a factor of 6 overall from the original DAS traces). The overall data rate achieved by our test machine for this phase was 602 KB/s for the original DAS traces versus 137 KB/s for the PDATS traces. This reflects the

greater efficiency of the 20 byte fixed-size `fread()` call used to read each record from the DAS file, versus the multiple small reads used to process the variable-length PDATS records.

Nevertheless, even in this case PDATS achieved a net speedup in reading the traces because of the reduced file sizes.

Access Time for ASCII Format Traces

In marked contrast to traces originally in binary format, PDATS produces large speedups in reading traces that were originally in ASCII format. This is due in large measure to the variable-length records in the ASCII format traces, as well as to the need to convert ASCII to integers.

For example, the set of Schieber format traces takes 9 times longer to read from the disk than the equivalent PDATS traces. We observed that the PDATS files take slightly longer to read than the uncompressed binary files, due to the absence of any significant spatial locality in these filtered traces. Again, the single fixed-size `fread()` used for the binary trace is sufficiently faster than the multiple reads for each PDATS record to overcome the 61% smaller PDATS files.

As an ASCII format with very good compression, the access time improvement for the Green Stamp trace was excellent, with a speedup of 27.5 from the original trace to the PDATS file.

The dinero format DLX traces exhibit qualitatively similar speedup results to the Green Stamp trace. The overall speedup is not as great (averaging 10.8) because the compression ratios were similarly smaller.

Finally, the VAX dinero traces exhibit essentially the same speedups as the DLX dinero traces, averaging 9.28.

Summary of Access Time Results

While little speedup was achieved relative to files already in binary form (DAS traces), the access time was improved by an order of magnitude for ASCII traces without time stamps, with even greater improvement for ASCII traces with time stamps.

Simulation Time Comparison

To gauge the effect of the improved access times resulting from PDATS on the speed of a well-known trace consuming program, we modified the `readfrominputstream()` function of the `dineroIII` cache simulator to call a function that extracts references from a PDATS trace. This modified program is called `dinerop`.

Because of the sophistication of the dinero simulator, a significant amount of processing is performed after each reference is read from the trace. This had the effect of diluting the access time speedups to a simulation speedup of 1.5.

CONCLUSIONS AND FUTURE WORK

Reduced storage requirements and shorter processing times are of obvious benefit to all users of traces. The PDATS technique described here provides both, while retaining all of the references of the original traces as well as any time stamps present.

Employed alone, PDATS achieves nearly order of magnitude improvements in storage space and access time, particularly for RISC processor traces. When Lempel-Ziv compression is applied after the trace-specific compression of PDATS, another half order of magnitude of compression can be obtained, although some of the access time improvement may be lost.

As a result of these findings, we plan to use PDATS as the standard trace format for a data base of traces being established at NMSU for the use of researchers and educators worldwide. (Our goal is to provide easy access to a large body of useful traces so that researchers will be able to reproduce and extend each others' work, a capability long enjoyed by researchers in the physical sciences.)

Planned enhancements to the PDATS technique include extensions to the file format for multiprocessor traces, and the possible inclusion of low-level I/O and a variant of Lempel-Ziv compression directly within the trace reading routine for improved speed.

Another possible application of the compression scheme is a hardware implementation to compress traces on the fly during real-time trace acquisition. When combined with some degree of trace filtering, it may be possible to reduce uniprocessor trace data bandwidth to that sustainable by a disk array, allowing the capture of extremely long complete traces in real time.

REFERENCES

1. Agarwal, A. and M. Huffman, "Blocking: Exploiting Spatial Locality for Trace Compaction," *Proceedings, ACM SIGMETRICS 1990*.
2. Agarwal, A., *et al.*, "ATUM: A New Technique for Capturing Address Traces Using Microcode," *Proceedings, 13th Annual International Symposium on Computer Architecture*: 119-127, 1986.
3. Das, S., "Effects of Line Size Variation on Cache Simulations Using Filtered Traces," Master's Thesis, New Mexico State University, 1993.
4. Hennessy, J.L. and D.A. Patterson, *Computer Architecture – A Quantitative Approach*, Morgan Kaufmann, San Mateo, CA, 1990.
5. Malik, N., *et al.*, "Architectural Effects on Dual Instruction Issue with Interlock Collapsing ALUs," *Proceedings, 1993 International Phoenix Conference on Computers and Communications*: 42-48.
6. Puzak, T.R., "Analysis of Cache Replacement Algorithms," Ph.D. Dissertation, University of Massachusetts, 1985.
7. Samples, A.D., "Mache: No-Loss Trace Compaction," *Proceedings, ACM SIGMETRICS 1989*: 89-97.
8. Schieber, C.D., "Real-Time Acquisition and Compression of Address Traces Using a Hardware Monitor," Master's Thesis, New Mexico State University, 1991.
9. Smith, A.J., "Two Methods for the Efficient Analysis of Memory Address Trace Data," *IEEE Transactions on Software Engineering*, **SE-3** (1), 1977.
10. Welch, T.A., "A Technique for High-Performance Data Compression," *IEEE Computer*, **17** (6): 8-19, 1984.