PSnAP: Accurate Synthetic Address Streams Through Memory Profiles

Catherine Mills Olschanowsky[1], Mustafa M. Tikir[2], Laura Carrington[2], and Allan Snavely[2]

[1] Department of Computer Science and Engineering
University of California at San Diego
cmills@cs.ucsd.edu
[2] San Diego Supercomputer Center
{mtikir,lcarring,allans}@sdsc.edu

Abstract. Memory address traces are an important information source; they drive memory simulations for performance modeling, systems design and application tuning. For long running applications, the direct use of an address trace is complicated by its size. Previous attempts to reduce address trace size incurred a substantial penalty with respect to trace accuracy. We propose a novel method of memory profiling that enables the generation of highly accurate synthetic traces with space requirements typically under 1% of the original traces. We demonstrate the synthetic trace accuracy in terms of cache hit rates, spatial-temporal locality scores and locality surfaces. Simulated cache hit rates from synthetic traces are within 3.5% of observed and on average are within 1.0% for L1 cache. Our profiles are on average 60 times smaller than compressed traces. The combination of small profile size and high similarity to original traces makes our technique uniquely applicable to performance modeling and trace driven simulation of large-scale parallel scientific applications.

1 Introduction

Trace-driven memory simulation is applicable to system design and evaluation, compilation (via trace-driven optimizations), and performance tuning. Today it is a standard practice to use address traces to explore the memory behavior of applications [1–5]. Simulation allows for the evaluation of new memory hierarchy designs without hardware implementation, this benefits both system design and evaluation for procurement. Modeling current workloads on proposed systems via simulation provides valuable insights, aiding in procurement decisions[6, 7]. Compiler optimization choices can be guided and evaluated through the examination and simulation of the resulting address streams. The accuracy and usefulness of each of these applications depends directly on the availability of relevant input, specifically relevant address traces.

Using traces from an actual anticipated scientific workload is the best policy for achieving accurate performance predictions and evaluations. The validity of a simulation driven study depends heavily on the chosen input workload; in the case of memory simulations the input is an address trace [8, 9]. VanderWiel [10] points out in a comparison study of two prefetching techniques, the performance improvement varied widely for each workload complicating the choice of prefetching technique. The performance results obtained by traces of small benchmarks chosen to represent a high performance computing (HPC) workload are of questionable relevance; choosing appropriate benchmarks is a difficult task, especially when applied to an HPC workload [11].

The direct collection and storage of full address traces is no longer practical due to the growth in the size of traces, driven by the increase in processor speed over the last three decades. Compounding this growth is the fact that HPC applications are scaling to larger and larger core counts where each processor produces a seperate stream of address requests at this rate. As a simple illustration, it is possible for a processor to issue more than a hundred million memory instructions per second. Assuming that each address is represented by 8 bytes, a full address trace grows by 800 million bytes a second, approximately 44GB a minute and 2.6TB an hour per processing core. Collecting an address trace for an application that runs several hours on thousands of processors is therefore not reasonable unless one leverages some regularity or recurring patterns in the application [12], but even with 90% compression the trace file sizes quickly become impractical [13].

Obtaining and storing relevant address traces is a fundamental requirement for trace-driven memory simulation of large parallel and HPC applications and the question must be asked: *how does one provide valid and relevant input of substantial size to a simulation?* Methods such as trace compression, truncation, on-the-fly processing, and synthetic trace generation have each been explored as an answer to this question. Each of the previously proposed solutions has shortcomings. Compression techniques incur a large slowdown [14, 15], and some of them require that the entire trace be stored before being compressed [14], truncating the trace loses valuable information. On-the-fly processing is done successfully, but uses a large amount of time on valuable HPC resources and has to be rerun each time the evaluation study changes [6]. Previous synthetic trace generation approaches have not reached high accuracy [16, 17].

A new method of address stream profile collection to be used in synthetic stream generation is presented in this paper. *PMaC Synthetic streams from address stream profiles*(PSnAP) offers accuracy at a granularity not before possible in synthetic trace generation. The size of the profiles is small enough that collecting them for an HPC application utilizing thousands of processors is possible.

Rather than taking a holistic view of an address trace as past attempts have, PSnAP breaks the trace down into two constituent parts, 1) program structure and 2) memory access pattern. PSnAP is able to capture both temporal and spatial locality characteristics as well as mimic fine-grained access patterns. Another important attribute of PSnAP is that the profiles are human readable and manipulatable.

There are several uses for the PSnAP streams. Almost any application for trace-driven performance analysis can potentially benefit from the ability to store and share memory streams. It is now possible to build a memory trace repository available to researchers for memory behavior research. Moreover, direct uses for the profiles are also possible. The profiles are small and human readable meaning that they can be manipulated in order to experiment with changing the behavior of the source application. This opens the possibility for automated code tuning and providing feedback to compilers on optimization decisions.

The unique contribution of this work is a method to summarize an address trace as it is generated, and to characterize it in a succinct and accurate fashion such that the result can be saved in a memory profile that can then be used to generate representative synthetic traces without going to the trouble of compiling, re-running and re-instrumenting the target application, as well as avoiding the space requirements for storing full address traces.

2 Methodology

PSnAP has two distinct phases 1) *capture* and 2) *replay*. During the capture phase an instrumented version of the application generates a compact profile that summarizes the important properties of the full application trace, using a binary rewriting tool, PMaCInst [18]. The replay phase, which can be done at any point in time after capture and does not require the use of an HPC system, uses the compact profile to generate a synthetic address trace that closely mimics the original trace.

A full application address trace can be viewed as a series of address traces resulting from the execution of loops which compose an application. In this work, a *stream profile* is a hierarchical representation of a full application address trace. Most scientific applications are composed of a series of loops. Through exploring the behavior within the constituent loops, we propose the application behavior is best characterized. Moreover, the address stream of each loop or *loop stream* can be viewed as accesses to disjoint regions in the memory. Figure 1 shows an application address stream for a pedagogical example consisting of a single loop. The loop stream is broken down into *memory region streams* as shown in the top right hand corner of the figure, where each region stream represents the accesses to a distinct area of memory.

To identify each loop stream within a full stream, the loops of an application are identified using static analysis and each basic block in the application is assigned to a loop. During the execution of instrumented executable (capture), each memory access is associated to a basic block which allows for mapping back to a loop. Basic blocks that appear within nested loops are assigned to the inner most loop.

For identifying the memory region streams within a loop stream, address membership in a memory region is determined using spatial characteristics. All of the addresses requested in an application refer to various areas of memory. Figure 1 shows how a simple loop's data structures may be laid out in memory. Variables *i*, *max*, and *total* are statically allocated and reside within close proximity to one another. Arrays *A* and *B* are each dynamically allocated and can be found separately in the heap. All the memory references which refer to a contiguous region of memory are referred to as memory region streams or just *regions*.

Each memory reference is assigned to a region by comparing its address to the range of addresses in the previously encountered regions. If the address falls within a (parameterized) distance to any of the addresses in these regions it is assumed to belong to the closest region[3]. Otherwise it is assumed that the address is part of a new region.

[3] The parameterized distance used for our experiments is 256 bytes, which was found to be adequate for accurate results for the applications we used.

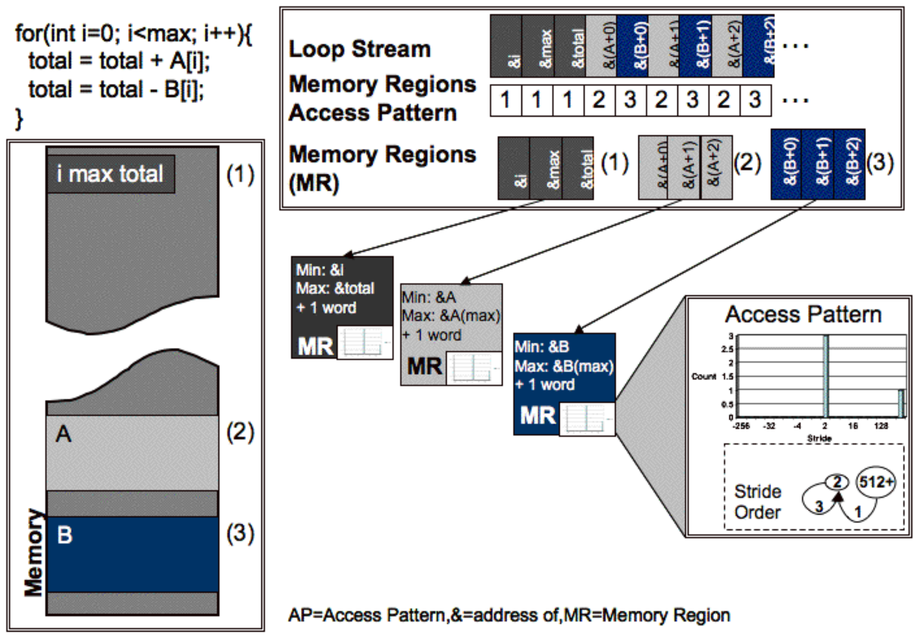

Fig. 1. An example of a loop stream broken down into a stream profile.

Once the regions have been defined each region stream is further characterized using three basic metrics: *access pattern*, *working set size* and *access count*. To characterize the access pattern of a region, a histogram of stride frequencies and a graph of stride ordering are maintained for each region. A stride is computed by comparing an incoming address to the one received immediately before, within the same region. Stride distances of 0 to 2^8 in increasing powers of 2 in both directions from the address are counted. The strides larger than 2^8 are counted as random; accesses with long strides and those with random srides tend to cause cache misses. The working set size of a region is identified by determining the minimum and maximum addresses encountered for that region. The number of addresses referring to each region is also stored.

In the case that a memory region is accessed in a random or incontiguous manner each access may result in the creation of a new region, preventing accurate profiling. For the loop streams that contain large strides or access an area of memory in a seemingly random manner (pointer chasing), we include a *merge* operation in our technique. The merge operation identifies groups of regions that have each been accessed a small number of times, generates a synthetic address stream representing those regions, and profiles the synthetic stream making sure to save all of the recorded information into a single memory region. This approach minimizes the number of regions and enables better working set size and access pattern identification.

A given loop stream is comprised of multiple region streams that are interleaved in a pattern that is stored in the pattern buffer. That pattern may be a simple alternating pattern as depicted in the pattern buffer in Figure 1 or it may be more complex and require a regular expression or function to express it. The current implementation uses a pattern buffer of a fixed size and simply saves the order that the regions are encountered until the buffer is full. [4]

The second phase, replay, is the process of synthesizing an address stream that can act as a representative proxy of the original. Each level of the hierarchically structured profile plays a part in the construction of the synthetic address stream for an application. The region metrics are used to generate addresses, the pattern buffers in the loop are used to interleave addresses and all of the loop streams are concatenated to create a full synthetic stream.

3 Results

In order to evaluate the effectiveness of PSnAP both the accuracy and efficiency are evaluated and compared to past work. The accuracy is measure using simulated cache hit rates and locality surfaces. PSnAP proves to be more accurate than any past attempts of lossy compression or synthetic trace generation. The size of the resulting profiles is shown to be small and a function of code complexity rather than execution time.

The evaluation uses a set of HPC benchmark kernals (listed in Table 2) and a set of memory hierarchies from recent HPC systems (listed in Table 1). The set of cache structures varies the three main cache characteristics: size, line size and associativity. The resulting cache structures are listed in Table 1. Structures one through three were chosen as modern examples of small, medium and large sized caches. Structures four and five are the Opteron and Budapest respectively, both are popular modern chips. Structure 6 is the cache structure used on the Power6 architecture from IBM and was chosen to represent the state-of-the-art in memory subsystem design. The remainder of the caches are variations on cache 3 with different line sizes and associativities. Our experiments show that the synthetic traces generated using our method are very accurate and the size requirements are extremely small.

3.1 Cache Simulation Results

The standard of accuracy measure for synthetic trace generation techniques is a comparison of cache simulation results between the synthetic stream and the original stream. Previously, the majority of cache simulation results have been presented using the cache hit rate average across the entire execution of a benchmark. It is well understood that as an execution proceeds, the cache hit rate of that execution changes dynamically. This may be due to changes in the code being executed (phases) or by changes over time due to data irregularity. Either way, estimating the similarity of two address streams over an entire execution may lead to error cancelation. Errors incurred during various program phases can cancel each other out causing the overall cache hit rates to appear more accurate than they really are. Hence, to investigate the accuracy of our approach, we have broken the execution and subsequent address streams down into sub streams; one stream per relevant loop or function as appropriate. This breakdown enables us to perform an accuracy comparison at a more granular level.

[4] Currently we use 1K accesses as the size of the pattern buffer.

ID	L1 Size (KB)	L1 Line (Bytes)	L1 Assoc.	L2 Size (KB)	L2 Line (Bytes)	L2 Assoc.	L3 Size (KB)	L3 Line (Bytes)	L3 Assoc.	Architecture
1	32	128	2	1024	128	8				PowerPC
2	256	128	8	9216	128	12				IT2
3	64	64	2	512	64	16				MIPS SiCortex
4	32	32	4	128	64	2				Opteron
5	64	64	2	512	64	16	1024	64	48	Budapest
6	64	128	8	4096	128	8	16384	128	16	IBM P6
7	64	64	2	512	64	8				
8	64	64	2	512	64	32				
9	64	64	2	512	32	16				
10	64	64	2	512	128	16				

Table 1. A summary of the cache structures used for cache hit rate accuracy verification.

Benchmark	Source	Average (%) Error L1	L2	L3
CG	NPB [19]	0.2	0.2	0.2
FT	NPB	0.1	0.1	0.1
Stream	HPCC	0.2	0.3	1.6
NBody	Aarseth Code [20]	1.8	1.2	1.6
Jacobi3D	Sci. Comp. at UCSD	2.7	3.0	3.4
HPL	HPCC	0.0	0.0	0.0

Table 2. A summary of the combined cache hit rate comparisons for benchmarks (All errors are averaged as absolute values).

Using an existing framework [18] the observed address stream of each benchmark was fed into a series of cache simulators. The cache simulations produce cache hit rates for each loop in the application. These cache hit rates are compared with the cache hit rates that result from the simulation driven by the synthetic generated streams.

Figure 2 presents the error between the cache hit rates for the observed and synthetic address streams for the most significant loop in each benchmark. The significance of a loop is determined by the number of memory operations that result from its full execution. The x-axis of the plots represent the different cache structures (the ids correspond to those in Table 1). Each figure shows the synthetic hit rate(blue stripes) and observed hit rate(solid yellow) as well as the absolute difference between the rates(black square). The figures representing L1 cache data have an addition piece of information, the estimated error that this synthetic stream could impose on a full performance execution time prediction(red asterisk). This is calculated using the basic equation for average memory access time found in Hennessy and Patterson [21]. The data point is only available for a subset of the cache structures.

Table 2 summarizes the error in cache hit rates averaged over all of the relevant loops of an application and all the cache structures used. The set of figures along with the table summarizing the cache hit rate errors for all benchmarks demonstrate very clearly that

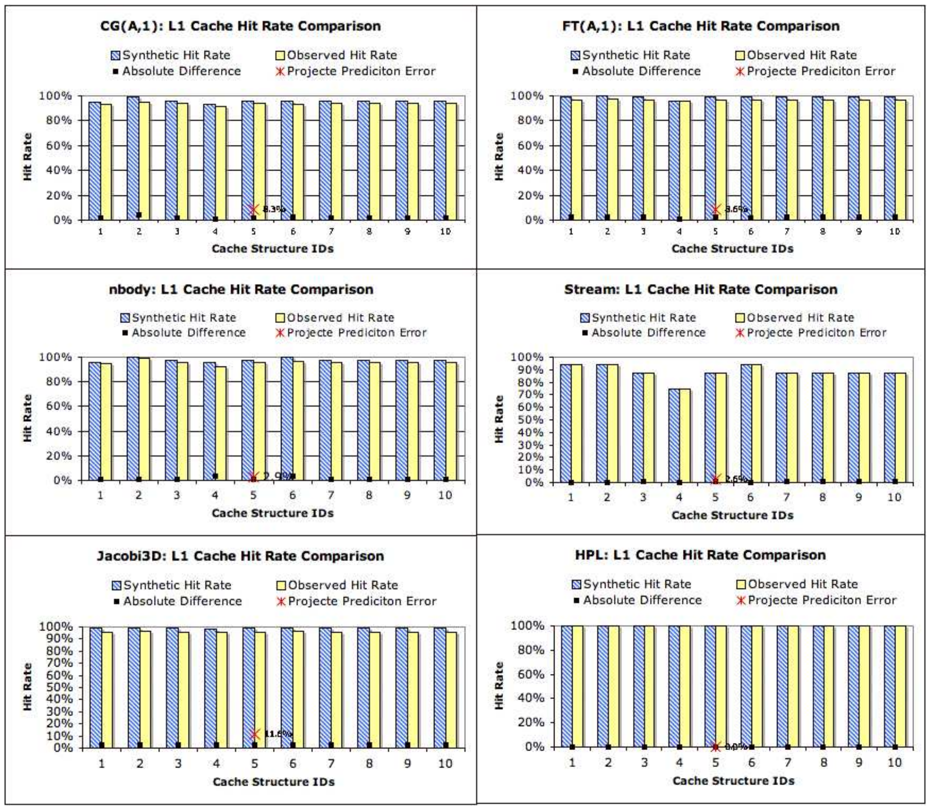

Fig. 2. Comparison of synthetic stream cache hit rates and rates for original stream across all 10 cache structures for the most dominant loop in each benchmark. Note that cache hit rates are high, as is expected of well-optimized HPC applications.

the synthetic streams are very similar to the observed in terms of performance. The error is consistently below 3%. The CG, FT, Stream and HPL benchmarks are almost perfectly reproduced with this method; Jacobi3D and Nbody both have much more complex access patterns, and are still well represented. The error in memory access time indicates a need for high accuracy, as any error in cache hit rate is multiplied in the full performance prediction. This shows that our approach is effective in generating synthetic traces that mimic the original trace.

3.2 Locality Surfaces

Another method of evaluating the accuracy of synthetic traces is to compare locality surfaces. Locality surfaces are one of the most effective ways to visualize the temporal and spatial locality characteristics of an address stream. Hence, by comparing the locality surfaces of a synthetically generated stream and its original counterpart, one can compare whether two streams exhibit the same locality behavior. If the locality surfaces look similar in shape, one can conclude the synthetic stream mimics the original one in terms of temporal and spatial locality.

We generate locality surfaces for both address streams for each benchmark. For locality surface generation, we used the implementation described by Grimsrud [17] and limited the field of the surface to strides within 256 bytes and distances within 64K. These limits still capture most of the interesting characteristics of the surface, and keep the overhead bearable (locality surfaces are notoriously expensive to construct). For our experiments, locality surfaces are generated on a per loop basis for the same reasons described above.

In Grimsrud [17], a locality surface is generated by tabulating a large histogram. Each address is compared to all of those that come after it until it is compared to itself. The bin in the histogram that corresponds to the stride and distance between each address is incremented during the comparison. The stride is the distances between the two memory addresses and the distance is the number of addresses that were encountered between them in the stream. The locality surface is a 3D representation of the histogram.

Grimsrud presents a discussion of how to interpret the characteristics of each surface [17]. The keys for comparison are that the same constructs appear in both surfaces and that their scale with respect to other constructs in the surface are similar. Key constructs are *sequential ridges* indicating a fixed stride through a data array and *decaying temporal ridge* indicating a value being reused over time.

Figure 3 presents a direct view of the locality surfaces for the benchmark CG for both the synthetic address stream(left) generated by our approach and the original stream(right). Figure 3 shows that the locality surface of the synthetic stream is very similar to original stream. In both surfaces the ratio of accesses with a stride between -2^3 and 2^4 are similar. The ridge down the center of the surface (a decaying temporal ridge) represents temporally repeated accesses and is present in both surfaces. The synthetic stream has smoothed the ridge out rather than reflecting the true behavior with two spikes. This can occur when the separate region streams have become out of synch with the pattern buffer. When this occurs the correct accesses are represented in the stream, but the distance between them may be skewed. It is interesting to note that the synthetic surface captures the sequential ridge, the ridge moving at a diagonal from the center. Strided accesses such as this have a large effect on performance and are therefore important to capture. PSnAP is able to reproduce the locality surfaces for loops with high accuracy for the synthetic streams indicating that our approach does not change the locality characteristics of the original stream and exhibits very similar behavior.

3.3 Comparison to Related Projects

Two categories of research warrant direct comparison to PSnAP: 1) trace compression algorithms and 2) synthetic trace generation methods. Lossless compression techniques have perfect accuracy at the expense of lower compression ratios and larger overhead times as compared to its lossy counterpart. Lossy compression algorithms represent an improvement in time and space overhead, with the addition of some inaccuracy.

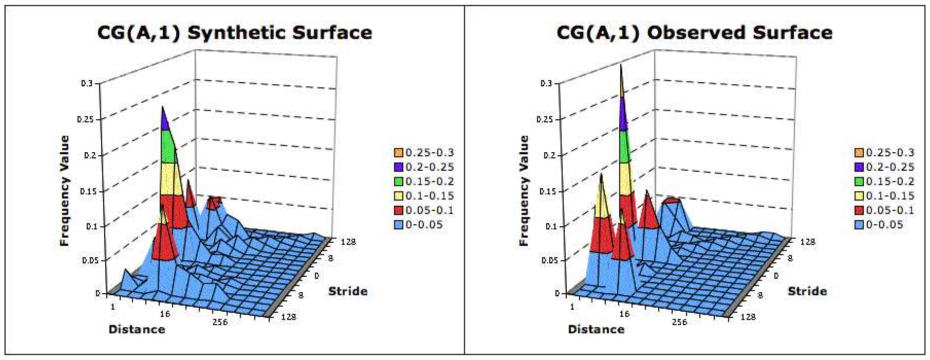

Fig. 3. The locality surfaces resulting from the observed and synthetic traces for the most influential loop in CG.A.1.

Benchmark	Full Trace Size	Stream/Profile Size		% Abs Err in Cache Hit Rates					
		PGGTC	PSnAP	PGGTC			PSnAP		
	(GB)	(KB)	(KB)	(L1)	(L2)	(L3)	(L1)	(L2)	(L3)
CG.A	5.4	22,620	153	1.5	1.0	0.2	0.2	0.1	0.03
EP.A	7.4	9,369	55	0.1	0.7	0.2	3.0	2.6	1.7
FT.A	18.7	1,129	317	2.9	1.6	0.3	0.1	0.1	0.05
IS.A	3.1	700	58	3.1	2.1	0.2	2.6	1.9	1.8
MG.A	12.6	5,033	324	3.8	2.9	0.8	1.1	0.9	0.6

Table 3. The compression achieved and the time required by PGGTC and our approach.

Sequitor [14] and Path Grammar Guided Trace Compression (PGGTC) [15] are both trace compression techniques developed specifically for address traces, they are lossless and lossy respectively. Both depend on the creation of a context free grammar (CFG) that represents repeated portions of the address trace. Sequitor creates the CFG dynamically and PGGTC creates the CFG using the control flow graph determined through static analysis of the application.

Table 3 presents a summary of the results for data compression accomplished using PGGTC for the NAS parallel benchmarks[5]. The data is extracted from Gao et al. [15]. It also includes the results of our approach in terms of the size of the stream profiles. This data shows that our approach has space requirements that is significantly smaller than PGGTC, on average 60X smaller.

[5] The measurements for CG and FT vary from table 2 to table 3. Two factors cause this discrepancy. First, table 2 uses data collected on benchmarks run across only a single processor versus table 3 that is run across four. Second, and more importantly the errors are calculated differently. In order to do a direct comparison with PGGTC the errors in table 3 are calculated using the difference between the average hit rate recorded over the entire address stream. The data in table 2 is calculated by averaging the absolute error across all of the significant sections of the stream, preventing any cancellation in error.

Table 3 also presents the percentage error between the hit rates for the original stream and the cache hit rates for the synthetic traces generated by both lossy portion of PGGTC and our approach. Table shows that hit rates for the traces generated by the lossy portion of PGGTC is similar to the hit rates of the traces generated by our approach. Both PGGTC and our approach maintained an error rate of less than 4% for L1 cache hit rates, 2% for L2 and 1% for L3 compared to the original address streams. Our approach performed slightly better than PGGTC for L1 caches.

Table 3 demonstrates that our approach is more effective compared to the compression techniques in two ways. First, the resulting size of the memory profiles is significantly smaller than the compressed traces by PGGTC. Second, the memory profiles are in a human readable format that enables them to be used to gain insight to the behavior of the application.

In a comparable area of synthetic trace generation, Weinburg [22] presented a synthetic trace generation tool called Chameleon. Chameleon is able to reproduce cache hit rates for a series of single level LRU caches for a sampling of address stream of the NAS parallel benchmarks. Using the same cache structures, our approach consistently resulted in a lower absolute error between the hit rates for actual traces and the synthetic traces generated. For IS.B.1 benchmark, Chameleon reported a maximum error of 30% in cache hit rates between the actual and synthetic trace whereas the maximum error for our method is around 10%.

Grimsrud [17], followed later by Sorenson [23], used locality surfaces and cache hit rates to measure the accuracy of five categories of synthetic address stream generation techniques. The conclusion drawn by both Grimsrud and Sorenson was that the synthetic trace generation techniques did not offer satisfactory accuracy with respect to representing the spatial and temporal locality characteristics of real traces.

In order to compare our synthetic trace generation technique with those evaluated by Sorenson, we implemented the described locality surface method and generated a surface for the same trace used in their comparison [16]. In order to match more closely the results found by Sorenson and Grimsrud, we used a trace obtained from *Twolf* from SPEC CPU2000 benchmark suite as the application. We used the address stream of the most important loop of Twolf to generate locality surfaces. This essentially zooms in the view of the surface and gives a higher level of detail. Moreover, the Twolf benchmark executes simulating annealing and produces a stream that is very difficult to summarize in a concise way.

Figure 4 presents the locality surfaces resulting from the observed address stream and the PSnAP synthetic stream. Figure 4 shows that the locality surfaces of original address stream and synthetic stream generated matches fairly in terms of its shape, especially for the most dominant part of lower stride-accesses. The most visible difference is the peak at stride two, distance two (in the middle by the back wall). PSnAP has moved some of the stride two references to a distance of four and overestimated the ratio of access with stride -16, making the ratio of accesses at the center peak shrink. This change, while visibly obvious, does not have a large affect on performance. Figure 4 demonstrates that the synthetic stream generated by our approach is able to maintain similar spatial and temporal locality behavior of the actual address stream.

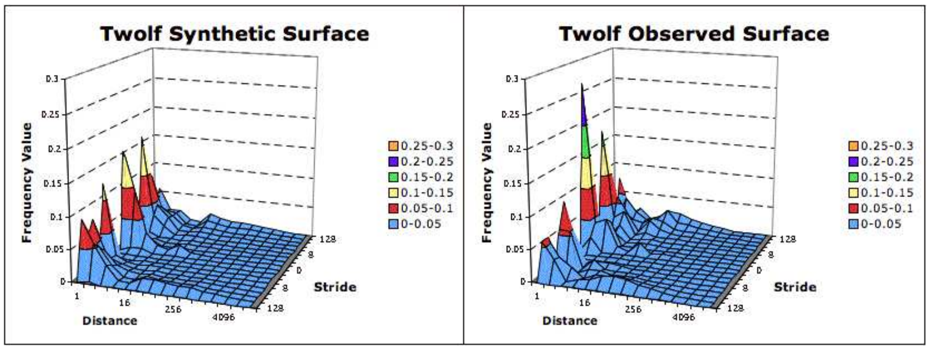

Fig. 4. Locality surfaces for Twolf for an observed stream and synthetic stream generated by our approach.

3.4 Size and Slowdown

The size and scaling behavior of the memory profiles are major advantages of the PSnAP approach. Each of the benchmarks used for the accuracy evaluation produced memory profiles of less than 250MB. This amount of data can easily be shared among collaborators. Even more interesting is that the profile size is not a function of execution time, but a function of code complexity.

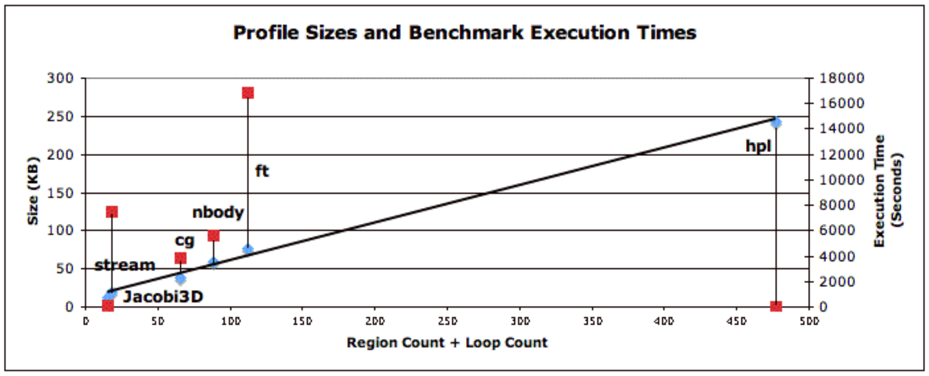

Fig. 5. The profile sizes and execution times of each benchmark plotted against a complexity measure.

We define the complexity measure(CM) of an application to be a combination of the number of loops and the number of distance memory regions used within those loops. The following equation shows how those code attributes are combined with attributes of the profile format: LoopCount and RegionCount are attributes of the code and the constants 1000,80, and 3610 represent the maximum number of bytes used by the pattern buffer, region histogram, and stride order graph respectively.

$$CM = (1000 * LoopCount + (80 + 3610) * RegionCount)/(1000 + 80 + 3610) \quad (1)$$

Figure 5 demonstrates that the profile sizes(blue dots corresponding to left axis) are a linear function of CM(x-axis). It is obvious that the corresponding execution times(red squares corresponding to right axis) are not dependent on the CM. The execution time is directly above or below the corresponding profile size.

The slowdown incurred during the instrumented runs is typical of binary instrumentation projects. The average observed slowdown is 169X (min: 7X max: 292X). This overhead presents a challenge for the use of this instrumentation, but it is important to note that the measurements were taken using the initial implementation of the tool and that performance improvements are expected. Possibilities for code optimization as well as sampling methods are being explored.

If we interpret these results in the light of the suitability of this work for capturing large address streams of long parallel running applications we note that, as to size, the worst case we experienced (HPL) was about 1 MB, a more than 100x compression over the raw address storage rate (you could store 1,000 processor's worth in 1 GB) and also note that this trace representation would NOT grow as a function of time but only as a function of complexity and different functions accessed during the program run (in the case of HPL it would not grow at all regardless of runtime). As to time, the slowdown may seem onerous for a long running program, is not beyond the realm of what in-depth performance studies may entail. For example, it is described in [24], how one million processor hours were used to characterize a strategic workload.

4 Related Work

Previous work in the area of synthetic stream generation covers a wide area of projects, some of which are described below. The independent reference model (IRM) [25, 26] profiles an execution to determine the frequency with which each page in the working set is accessed. A synthetic stream is then generated that contains the same frequency for each page. The accuracy resulting from this method is not high enough because it models each page independently and important patterns due to locality are lost. Weinberg[22] applied a modified version of IRM that recorded the probability of accessing an area of memory using a tree structure that represented increasingly smaller areas of memory. This model also suffers from inaccuracy due to an inability to preserve key patterns in the address stream, especially regular strided accesses resulting from loop constructs, a characteristic we are able to preserve.

The distance model[27] models the probability of specific distances between neighboring addresses rather than modeling the frequency of the appearance of the addresses themselves. Thiebaut et al[28] extended the distance model using a hyperbolic probability function to model the size of the steps between references. This approach can maintain some of the statistical properties of the stream, but the underlying patterns are lost.

Agarwal[29] suggested the Partial Markov Model (PMM). This model depends on a two state Markov chain, where state 0 produces strided addresses and state 1 produces random. The state transitions are controlled by a probability function. This model is not able to capture relevant temporal locality traits of the address stream and does not capture the behavior of two strided streams being called in turn, a very common pattern.

Berg[30] modeled the reuse distance between addresses using a probability function. The reuse distance is the number of addresses accessed between accesses to the same address. Recording reuse distance during tracing is an expensive operation and impractical for large scientific applications. The stack distance model[2, 31] maintains an ordered list of encountered addresses and models the probability of accessing an address some distance from the top of the list or stack. It is related to the reuse distance, because the stack distance is the number of *unique* references which appear between accesses to the same reference. Hassan[32] extends the stack model with the edition of a Markov chain and generates synthetic traces for the purpose of driving trace-driven simulations of cache memory. This approach is the most accurate of the presented projects, but results are only presented for single level LRU caches, whereas our approach is shown to be accurate on a large collection of multi-level realistic cache structures. Tracing overhead time and space requirements are also not presented, preventing an in depth comparison.

Grimsrud[17] and later Sorenson[16] evaluated the accuracy of several address stream models using locality surfaces. The surfaces are able to capture both spatial and temporal locality characteristics. We apply similar surfaces to our synthetics streams, however, they are applied to portions of the execution rather than the entirety in order to demonstrate that the behavior of applications changes over time.

All of the above attempt to describe the address stream of an application in a holistic manner. We are able to achieve a higher level of accuracy and maintain complex patterns in the streams, which prove important for simulation driven analysis.

The stack distance model, mentioned above, was used by Cascaval et. al [33] to perform compile-time based performance predictions. This application of the stack distance model has no requirement for address trace storage, but this work may be complementary in that the PSnaP profiles can be partially generated from compile-time statistics and yields higher accuracy than the stack distance model.

5 Conclusion

We present a method of creating a compact profile of an application to generate accurate synthetic traces for the application. The profiles are a compact and succinct summary of a full address streams, more compact than any previous approach. In this method, rather than taking a holistic view of an address trace as previous attempts have, a full trace of an application is broken down into constituent parts using the program structure and memory access patterns.

We evaluate the accuracy of synthetic traces by comparing their cache hit rates and locality surfaces to those of observed traces. Our experiments demonstrate that PSnAP synthetic traces closely mimic the observed address traces of applications in terms of cache-ability. The average error between the hit rates for synthetic and original traces is 2.2% for L1 caches, 1.9% for L2 caches and 1.8% for L3 caches. More importantly, the locality surfaces for synthetic traces match the locality surfaces for the observed traces indicating that our approach exhibits the same locality characteristics of the observed streams.

We demonstrate that highly accurate synthetic traces can be generated from very compact stream profiles. This combination of traits makes this method uniquely suitable for performance modeling of large-scale scientific HPC workloads. Due to the characteristic that the stream profiles' size scales with code complexity rather than runtime, it is possible to collect a stream profile for even long running parallel applications.

References

1. K. Skadron, M. Martonosi, D. I. August, M. D. Hill, D. J. Lilja, and V. S. Pai, "Challenges in computer architecture evaluation," *Computer*, vol. 36, no. 8, pp. 30–36, 2003.
2. R. Mattson, J. Gecsei, D. Slutz, and I. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, pp. 78 – 117, 1970.
3. P. Calingaert, "System performance evaluation: survey and appraisal," *Commun. ACM*, vol. 10, no. 1, pp. 12–18, 1967.
4. W. Anacker and C. P. Wang, "Evaluation of computing systems with memory hierarchies," *IEEE Transactions on Electronic Computers*, vol. EC-16, no. 6, pp. 670–679, December 1967.
5. W. Anacker and C. Wang, "Performance evaluation of computing systems with memory hierarchies," *Electronic Computers, IEEE Transactions on*, vol. EC-16, no. 6, pp. 764–773, Dec. 1967.
6. A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for application performance modeling and prediction," in *ACM/IEEE Conference on High Performance Networking and Computing*, 2002.
7. L.Carrington, N.Wolter, A.Snavely, and C.Lee, "Applying an automated framework to produce accurate blind performance predictions of full-scale hpc applications," in *UGC*, 2004.
8. J. Flanagan, B. Nelson, and G. Thompson, "The inaccuracy of trace-driven simulation using incomplete multiprogramming trace data," in *MASCOTS*, 1996.
9. D. R. Kaeli, "Issues in trace-driven simulation," in *Performance Evaluation of Computer and Communication Systems, Joint Tutorial Papers of Performance '93 and Sigmetrics '93*. London, UK: Springer-Verlag, 1993, pp. 224–244.
10. S. P. Vanderwiel and D. J. Lilja, "Data prefetch mechanisms," *ACM Comput. Surv.*, vol. 32, no. 2, pp. 174–199, 2000.
11. R. C. Murphy and P. M. Kogge, "On the memory access patterns of supercomputer applications: Benchmark selection and its implications," *IEEE Trans. Comput.*, vol. 56, no. 7, pp. 937–945, 2007.
12. M. Laurenzano, B. Simon, A. Snavely, and M. Gunn, "Low cost trace-driven memory simulation using simpoint," in *Workshop on Binary Instrumentation and Applications*, 2005.
13. X. Gao, "Reducing time and space costs of memory tracing," Ph.D. dissertation, University of California at San Diego, La Jolla, CA, USA, 2006.
14. S. Mitarai, M. Hirao, T. Matsumoto, A. Shinohara, M. Takeda, and S. Arikawa, "Compressed pattern matching for SEQUITUR," in *Data Compression Conference*, 2001, pp. 469+.
15. X. Gao, A. Snavely, and L. Carter, "Path grammar guided trace compression and trace approximation," in *The 15th IEEE International Symposium on High Performance Distributed Computing*, 2006.
16. E. Sorenson and J. Flanagan, "Evaluating synthetic trace models using locality surfaces," *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on*, pp. 23–33, Nov. 2002.

17. K. Grimsrud, J. Archibald, R. Frost, and B. Nelson, "On the accuracy of memory reference models," in *Proceedings of the 7th international conference on Computer performance evaluation : modelling techniques and tools*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1994, pp. 369–388.
18. M. Tikir, M. Laurenzano, L. Carrington, and A. Snavely, "The pmac binary instrumentation library for powerpc," in *Workshop on Binary Instrumentation and Applications*, 2006.
19. R. C. Agarwal, B. Alpern, L. Carter, F. G. Gustavson, D. J. Klepacki, R. Lawrence, and M. Zubair, "High-performance parallel implementations of the NAS kernel benchmarks on the IBM sp2," *IBM Systems Journal*, vol. 34, no. 2, pp. 263–272, 1995.
20. S. Aarseth, "Nbody2: a direct n-body integration code," *New Astronomy*, vol. 6, p. 277, 2001.
21. J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Third, Ed. Morgan Kaufmann, 2003.
22. J. Weinberg and A. Snavely, "Chameleon: A framework for observing, understanding, and imitating memory behavior," in *PARA08: Workshop on State-of-the-Art in Scientific and Parallel Computing*, Trondheim, Norway, May 2008.
23. E. S. Sorenson and J. K. Flanagan, "Using locality surfaces to characterize the specint 2000 benchmark suite," in *In Lizy Kurian John and Ann Marie Grizza Maynard, editors, Workload Characterization of Emerging Computer Applications*. Kluwer Academic Publishers, 2001, pp. 101–120.
24. X. Gao, M. Laurenzano, B. Simon, and A. Snavely, "Reducing overheads for acquiring dynamic traces," in *International Symposium on Workload Characterization*, 2005.
25. P. J. Denning, "On modeling program behavior," in *AFIPS '71 (Fall): Proceedings of the November 16-18, 1971, fall joint computer conference*. New York, NY, USA: ACM, 1971, pp. 937–944.
26. A. V. Aho, P. J. Denning, and J. D. Ullman, "Principles of optimal page replacement," *J. ACM*, vol. 18, no. 1, pp. 80–93, 1971.
27. J. Spirn, "Distance string models for program behavior," *Computer*, vol. 9, no. 11, pp. 14–20, Nov. 1976.
28. D. Thiebaut, J. Wolf, and H. Stone, "Synthetic traces for trace-driven simulation of cache memories," *IEEE Transactions on Computers*, vol. 41, no. 4, pp. 388–410, 1992.
29. A. Agarwal, J. Hennessy, and M. Horowitz, "An analytical cache model," *ACM Trans. Comput. Syst.*, vol. 7, no. 2, pp. 184–215, 1989.
30. E. Berg and E. Hagersten, "Statcache: a probabilistic approach to efficient and accurate data locality analysis," in *ISPASS '04: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 20–27.
31. J. Archibald and J.-L. Baer, "Cache coherence protocols: evaluation using a multiprocessor simulation model," *ACM Trans. Comput. Syst.*, vol. 4, no. 4, pp. 273–298, 1986.
32. R. Hassan, A. Harris, N. Topham, and A. Efthymiou, "Synthetic trace-driven simulation of cache memory," *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, vol. 1, pp. 764–771, May 2007.
33. C. Cascaval, L. DeRose, D. A. Padua, and D. A. Reed, "Compile-time based performance prediction," in *In Twelfth International Workshop on Languages and Compilers for Parallel Computing*, 1999, pp. 365–379.