Profile of AVUS Based on Sampled Memory Tracing of Basic Blocks

Laura C. Carrington, Xiaofeng Gao, & Allan Snavely

Performance Modeling and Characterization Lab
San Diego Supercomputer Center
University of California
San Diego, CA 92093
{lcarring,xgao,allans}@sdsc.edu

Roy L. Campbell, Jr.

High Performance Computing Division
Army Research Laboratory
Major Shared Resource Center
Aberdeen Proving Ground, MD 21005
rcampbell@arl.army.mil

Abstract

The latest PMaC performance prediction framework has been augmented to reveal the predicted memory execution time for each basic block of a target code, so that "hot spots" for memory-limited applications can be identified. This paper provides a discussion of the underlying methodology as well as a memory-based profile of AVUS. AVUS is a CFD code developed by the Air Force Research Laboratory to determine the non-linear response of a structure in a fluid.

1. Introduction.

Processor innovations have outpaced those of memory subsystems over an extended period of time and have resulted in an increasing number of codes that are limited by memory subsystem performance rather than the floating-point instruction rate of a processor. Therefore, users and code developers need a new tool to analyze such codes – a memory-based profiler that ranks basic blocks according to projected memory execution time and provides vital details about each basic block's use of the memory hierarchy.

To reveal the "hot spots" of a target code, some profilers rank subroutines in descending order according to processor-oriented metrics such as CPU cycles required, wall-clock time, or FLOP count. CPU cycles and FLOP count can obviously be misleading metrics for memory-limited codes. Wall-clock time, although it does indeed reveal those subroutines that consume the most time, does not reveal a detailed description of memory usage per subroutine. In fact, profiling at the subroutine level is often not fine-grained enough to provide useful information for memory-limited codes, since memory bottlenecks often arise from small nuances that comprise only a small fraction of a subroutine's code. Profiling at such a fine-grained level, normally requires hand instrumentation of the source code and can often prevent the compiler from fully optimizing the code, resulting in a very misleading profile.

This work uses MetaSim Profiler (a fine-grained memory-based profiling tool designed by the PMaC Lab) to profile a key computational fluid dynamics (CFD) code – AVUS – found in the DoD HPCMP TI-05 Benchmarking Suite. This profiling tool is closely associated with the PMaC predictive framework used for performance prediction of HPCMP codes on arbitrary systems. As a validation of the framework, performance prediction was previously applied to five TI-05 application test cases and 12 HPCMP systems, yielding an overall average absolute error of 18% [1]. Therefore, predicted memory execution time rankings are considered to be reflective of observed memory execution time rankings to the extent that the framework is accurate.

Data is gathered by MetaSim Profiling by tapping into two of the predictive framework's gathered traces – one pertaining to memory operations and the other pertaining to floating-point operations. The former trace is directly related to the desired memory profiling, and the latter is sought for purposes of comparing basic block rankings (i.e., memory versus floating point). These traces are fine-grained (i.e., have fundamental units of basic blocks), and are a result of automated instrumentation of the target code's binary, thereby preventing any impact to compiler optimization.

For this paper, two problem sizes are investigated (AVUS Standard and Large) for three different metrics: (1) number of FLOPs, (2) number of memory references, and (3) predicted memory execution time. The underlying methodology is described in Section 2, and results are presented in Section 3.

2. Methodology.

The memory-based profiler presented in this paper is a modification of the MetaSim Convolver used in the PMaC performance modeling framework. Details of this

framework can be found in [1], but a short description is provided below.

As shown in Figure 1, the framework consists of two main components: a single-processor model and a communication model. The single-processor model captures the behavior of a target application during periods of non-communication, while the communication model observes and records the specifics of that application's communication events. Each model consists of (1) traces that determine the basic operations to be performed by an application, irrespective of the target system (i.e., the application signature) and (2) corresponding measured rates for basic operations on a target system (i.e., the system profile). The application signature and the system profile of the target system are combined via convolution for both the single-processor and communication models and the results are then convolved to produce a performance prediction.

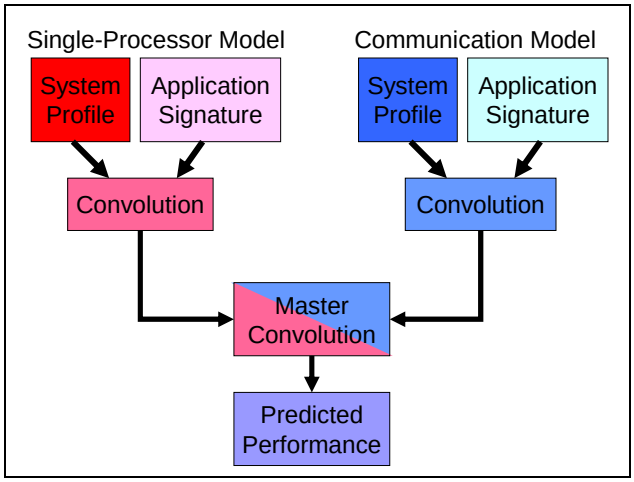

Figure 1. Diagram of PMaC performance prediction framework.

Traces for a single-processor model consist of memory and floating-point operation traces performed by the MetaSim Tracer. A memory operation trace contains information about the type of memory operations (i.e., stride-one or random) performed by the target application, and is gathered by instrumenting the application's binary via an automated binary rewriting process. To save disk space, MetaSim Tracer performs cache simulations while gathering memory traces. So rather than storing the entire memory address stream for an application execution, the address stream is piped through a cache simulator to obtain expected cache hit rates for a predefined set of target systems (i.e., those systems for which predictions are needed). Therefore, (per target application and system) a summary per basic-block of the expected cache hit rates, memory access patterns, and memory, floating-point, and branch operation counts is obtained.

The MetaSim Dependency Analyzer uses static analysis to examine the application binary (previously augmented by MetaSim Tracer) to determine the floating-point dependency and branch intensity of each basic-block. Data from both the MetaSim Dependency Analyzer and the MetaSim Tracer along with the system profile of the target system are examined by the MetaSim Convolver to determine a relative single-processor performance for the application between communication events on the target system.

MetaSim Convolver predicts overall performance based on estimates of memory and floating-point performance. Since, for the majority of applications, the CPU is not the dominate performance bottleneck, the following operational description will focus on the performance of the memory subsystem. To calculate memory performance for each basic-block, the convolver analyzes the trace information obtained by MetaSim Tracer and creates a table similar to Table 1. (Table 1 contains a small fraction of the information gathered by MetaSim Tracer.)

Table 1. Sample output for AVUS Large at 384 processors on the IBM P3 using MetaSim Tracer.

BB #[1]	MEM REF[2]	% RAND[3]	FPDP[4]	BDP[5]	Cache hit rates (L1,L2)	MEM BW[6] (MB/s)
7327		5			91,94	332
12963		19			93,97	64
10259		0			100,100	5250
10505		20			99,100	795

1. BB # = basic-block number
2. MEM REF = number of memory references
3. % RAND = percent of memory operations that are random stride
4. FPDP = floating-point dependency factor
5. BDP = branch dependency factor
6. MEM BW = memory bandwidth

The floating-point dependency factor is the average weighted distance between a value being created and its use. The branch dependency factor is the ratio of the number of branch operations to all other instructions in an inner loop. The memory bandwidth mentioned in the final column indicates the rate at which the basic-block is expected to perform based on its location in the memory subsystem (cache versus main memory) and access pattern (unit versus random stride). Figure 2 demonstrates how these values are selected from the system profile of the target system, and Equation 1 illustrates how memory time is calculated for each basic-block based on memory references, memory bandwidth, and dependency penalty.

$$\text{Memory Time} = \frac{\#\,\text{Mem Ref}}{\text{Mem BW}} \times \text{Dep Penalty} \quad (1)$$

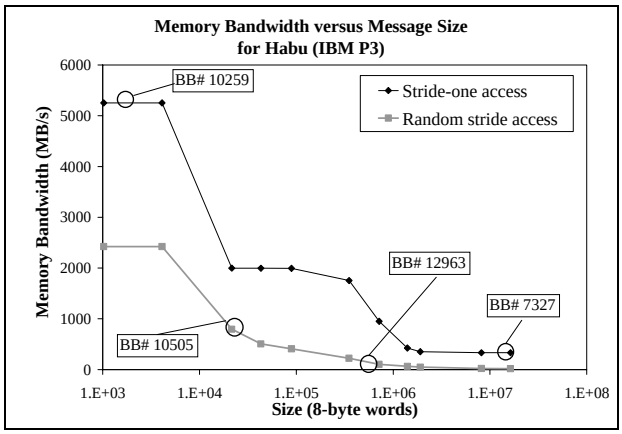

Figure 2. Memory bandwidth of IBM P3 at different message sizes for unit and random-stride access.

The floating-point dependency factor (FPDP) and branch dependency factor (BDP) are used as cutoff values in determining whether or not a basic-block's performance should be penalized due to its amount of dependency. The magnitude of the penalty is determined from benchmark probes that ascertain the target system's memory bandwidth for varying message size given the presence of floating-point and branch dependencies. Quantitatively, the dependency penalty is the ratio of the memory bandwidth with and without dependencies in the inner loop.

MetaSim Profiler is a modification to MetaSim Convolver, which provides detailed information for each basic-block to aid in code optimization. Table 2 presents a sample of the data provided.

Table 2. Sample output for AVUS Large at 384 processors on the IBM P3 using MetaSim Profiler.

BB #	MEM REF	FP OPs	Cache hit rates (L1,L2)	MMS1[1]	MPC[2]	MPCD[3]
10415	1.8E10	2.E10	93,98	433	433	433
6556	1.3E10	0	100,100	315	20	111
10288	4.7E9	3.E9	95,96	112	112	256
10484	3.8E9	3.E9	93,98	91	1588	1588

1. MMS1 = memory time assuming all memory references are performed in unit stride
2. MPC = memory time accounting for location in memory subsystem and access pattern
3. MPCD = same as MPC but additionally accounting for dependency

Each row represents one basic block of AVUS, executed using the large test case and 384 processors, while simulating the cache structure of the IBM P3. The first column provides the basic block number that can be mapped back to a line number and subroutine in the AVUS source code. The next two columns provide the total memory references and total floating-point operations performed. The fourth column provides the expected cache hit rates determined by the cache simulator. The fifth column, MMS1, provides a predicted "peak" memory time (i.e., the memory time assuming each reference performs at the speed of a unit-stride access from main memory). Although a system's true peak memory performance corresponds to a unit-stride access from L1 cache (also provided by the profiler, but not shown), the main memory bound gives more insight into performance bottlenecks when comparing it to other calculated memory times. The sixth column, MPC, provides the calculated memory time based on stride access and data locality of the basic-block. More specifically, for basic-blocks possessing an access pattern with more than 15% random-stride, all memory references (for that basic-block) are considered to perform at the random-stride access bandwidth; otherwise, unit-stride performance is assumed. If hit rates for the basic block are sufficiently high, either the unit-stride or random-stride memory performance curve can be used to determine performance (since the plateaus of these curves are meant to represent the performance at different levels of cache). The final column, MPCD, provides the calculated memory time based on memory access pattern, data locality, and dependency of the current basic block. Basic-blocks with a floating-point dependency or high branch intensity are penalized based on the system's ability to accommodate such factors (as captured by the benchmark probes).

The information provided in Table 2 can aid in the optimization of a particular basic-block through the comparison of its calculated memory times. For example, for BB 10415, MMS1 and MPC have the same value, indicating that this block performs as a unit-stride access out of main memory would. Since "peak" time is based on access to main memory, not cache, one optimization that could be investigated would be blocking the loop to allow for more reuse, thereby pushing the data set into cache. For BB 6556, data is clearly resident in cache, since MPC and MPCD are much smaller than MMS1; however, dependency is prohibiting the full benefit of cache performance, since MPCD is greater than MPC. For BB 10288, data is not resident in cache, and performance is affected by dependency, since MMS1 and MPC are the same and MPCD is greater than MPC. For BB 10484, data is not resident in cache and is fairly random in stride (since MPC is greater than MMS1), while performance is not affected by dependency (since MPC and MPCD are the same).

3. Analysis.

In addition to analyzing memory performance for a particular basic-block, the "hot spots" of an execution can be determined by ranking all basic-blocks according to a particular metric. For this paper, the basic blocks for the standard and large test cases of AVUS for three

processor counts each were ranked according to (1) the number of floating-point operations, (2) the number of memory references, and (3) MPCD (the calculated memory time based on memory access pattern, data locality, and dependency) for two different simulated cache structures – that of the IBM P3 and the IBM p655+. Each case was executed via the MetaSim Profiler to obtain data similar to that provided in Table 2. For the performance prediction of all 12 examined cases, the aggregate average absolute difference was 7%; therefore, MPCD and observed memory time are assumed to be interchangeable for the purposes of this paper.

Tables 3-14 provide comparative rankings for each case, revealing how the top 10 basic-blocks in terms of memory time ranked in terms of memory reference count (MRC) and floating point operation count (FPOC). Ranking by FPOC is better correlated to ranking by memory time for AVUS Standard on the p655+, but provides little to no advantage over ranking by MRC for the P3. On the other hand, ranking by MRC is better correlated to ranking by memory time for AVUS Large on the P3, but provides little to no advantage over ranking by FPOC for the p655+. In general, neither ranking by FPOC or ranking by MRC are well correlated to ranking by memory time, hence the need for a fine-grained profiling tool based primarily on memory time for memory-limited codes.

Table 15 (placed at the end of the paper due to its size) provides the top 10 basic-blocks in terms of memory time for all 12 cases. The top 10 are similar for all cases of a particular system, but are noticeably different between systems (due to the accounting of the memory hierarchy of each system). An exception would be AVUS Large at 128 processors in which case the memory time basic-block rankings for the P3 and the p655+ are very similar.

Basic blocks 10440, 10484, 19697, 20285, and 6553 are each in the memory time top two for at least one of the 12 cases. Block 10440 is always in the top two, and blocks 10440, 10448, and 19697 are always in the top 10 for all cases. Therefore, the performance of some portions of AVUS are insensitive to test case, processor count, and system and present consistent performance bottlenecks, while the performances of other portions are susceptible to context and may or may not be an issue for a particular execution.

Future work will associate each of these high-ranking basic blocks with corresponding code within AVUS, and will carefully analyze these isolated code sections to determine the specific aspects of each that engender a large memory time. Dynamic and static performance characteristics will also be examined with respect to execution context to further application insights.

Table 3. Comparative rankings for AVUS Standard and the IBM p655+ at 32 processors.

Memory References	Floating Point Operations	Memory Time
3	3	1
19	16	2
20	20	3
23	22	4
24	21	5
2	2	6
1	1	7
72	61	8
4	6	9
76	53	10

Table 4. Comparative rankings for AVUS Standard and the IBM p655+ at 64 processors.

Memory References	Floating Point Operations	Memory Time
3	3	1
19	16	2
20	20	3
23	22	4
24	21	5
2	2	6
1	1	7
74	62	8
83	53	9
86	47	10

Table 5. Comparative rankings for AVUS Standard and the IBM p655+ at 128 processors.

Memory References	Floating Point Operations	Memory Time
5	4	1
3	3	2
20	20	3
23	22	4
24	21	5
2	2	6
1	1	7
19	16	8
77	62	9
86	47	10

Table 6. Comparative rankings for AVUS Large and the IBM p655+ at 128 processors.

Memory References	Floating Point Operations	Memory Time
1	1	1
15	13	2
16	17	3
17	75	4
20	19	5
21	18	6
2	3	7
75	58	8
3	2	9
82	43	10

Table 7. Comparative rankings for AVUS Large and the IBM p655+ at 256 processors.

Memory References	Floating Point Operations	Memory Time
3	4	1
19	16	2
20	20	3
21	62	4
24	22	5
25	21	6
2	2	7
1	1	8
79	59	9
4	6	10

Table 8. Comparative rankings for AVUS Large and the IBM p655+ at 384 processors.

Memory References	Floating Point Operations	Memory Time
3	3	1
20	20	2
21	63	3
24	22	4
25	21	5
2	2	6
46	34	7
50	32	8
1	1	9
19	16	10

Table 9. Comparative rankings for AVUS Standard and the IBM P3 at 32 processors.

Memory References	Floating Point Operations	Memory Time
3	3	1
1	1	2
19	16	3
20	20	4
2	2	5
23	22	6
24	21	7
4	6	8
5	5	9
6	8	10

Table 10. Comparative rankings for AVUS Standard and the IBM P3 at 64 processors.

Memory References	Floating Point Operations	Memory Time
3	3	1
1	1	2
19	16	3
20	20	4
2	2	5
23	22	6
24	21	7
4	6	8
5	5	9
6	8	10

Table 11. Comparative rankings for AVUS Standard and the IBM P3 at 128 processors.

Memory References	Floating Point Operations	Memory Time
3	3	1
5	4	2
1	1	3
19	16	4
20	20	5
2	2	6
23	22	7
24	21	8
4	6	9
6	8	10

Table 12. Comparative rankings for AVUS Large and the IBM P3 at 128 processors.

Memory References	Floating Point Operations	Memory Time
1	1	1
15	13	2
16	17	3
17	156	4
20	19	5
21	18	6
2	3	7
3	2	8
4	5	9
5	4	10

Table 13. Comparative rankings for AVUS Large and the IBM P3 at 256 processors.

Memory References	Floating Point Operations	Memory Time
3	4	1
1	1	2
2	2	3
19	16	4
20	20	5
21	76	6
24	22	7
25	21	8
4	6	9
5	5	10

Table 14. Comparative rankings for AVUS Large and the IBM P3 at 384 processors.

Memory References	Floating Point Operations	Memory Time
3	3	1
1	1	2
2	2	3
19	16	4
20	20	5
21	73	6
24	22	7
25	21	8
4	5	9
5	4	10

4. Conclusions.

Fine-grained, memory time-to-solution analysis is paramount to understanding the performance of memory-limited codes. Accordingly, MetaSim Profiler was developed and used to identify key "hot-spots" in AVUS for different execution scenarios. In future work, code corresponding to each "hot-spot" will be carefully examined to determine if general as well as scenario-specific performance improvements can be achieved.

References

[1] Carrington, Gao, Wolter, Snavely, and Campbell, "Peformance sensitivity studies for strategic applications", Proceedings of the 2005 DoD HPCMP Users Group Conference, Nashville, TN, 2005.

Table 15. Top 10 basic-blocks in terms of memory time for AVUS Standard and Large using the IBM p655+ and IBM P3.

	AVUS Standard						AVUS Large					
	p655+			P3			p655+			P3		
Memory Time Rank	32p	64p	128p	32p	64p	128p	128p	256p	384p	128p	256p	384p
1	10440	10440	20285	10440	10440	10440	10440	10440	10440	10440	10440	10440
2	10484	10484	10440	6553	6553	20285	10484	10484	19697	10484	6553	6553
3	19697	19697	19697	10484	10484	6553	19697	19697	20151	19697	20158	20158
4	18808	18808	18808	19697	19697	10484	20151	20151	18808	20151	10484	10484
5	20324	20324	20324	20155	20155	19697	18808	18808	20324	18808	19697	19697
6	20155	20155	20155	18808	18808	20155	20324	20324	20158	20324	20151	20151
7	6553	6553	6553	20324	20324	18808	20163	20158	15402	20163	18808	18808
8	10504	10504	10484	10415	10415	20324	10504	6553	15393	20153	20324	20324
9	10415	12963	10504	20285	20285	10415	20153	10504	6553	10415	20163	20163
10	12963	20223	20223	20163	20163	20163	20223	20163	10484	20285	20153	20153