Workload Characterization: Motivation, Goals and Methodology

Lizy Kurian John, Purnima Vasudevan and Jyotsna Sabarinathan*
Electrical and Computer Engineering Department
The University of Texas at Austin
{ljohn,purnima,sabarina}@ece.utexas.edu

Abstract

Understanding the characteristics of workloads is extremely important in the design of efficient computer architectures. Accurate characterization of workload behavior leads to the design of improved architectures. The characterization of applications allows to tune processor micro-architecture, memory hierarchy and system architecture to suit particular features in programs. Workload characterization also has a significant impact on performance evaluation. Understanding the nature of the workload and its intrinsic features can help to interpret performance measurements and simulation results. Identifying and characterizing the intrinsic properties of an application in terms of its memory access behavior, locality, control flow behavior, instruction level parallelism, etc. can eventually lead to a program behavior model, which can be used in conjunction with a processor model to do analytical performance modeling of computer systems. In this paper, we describe the objectives of workload characterization and emphasize the importance of obtaining architecture independent metrics for workloads. A study of memory reference locality using some generic metrics is presented as an example.

1. Introduction

Understanding the characteristics of applications allows to tune processor micro-architecture, memory hierarchy and system architecture to suit particular features in programs. Workload characterization has a significant impact on performance evaluation. Understanding the nature of the workload and its intrinsic features can help to interpret performance measurements and simulation results better. For instance, data and control dependency related properties of programs can be used to draw deeper insight into the working of branch predictors and processor pipelines [Em86] [Ev98]. Interpreting performance evaluations using workload features is a popular use of workload characterization. Knowing what features in applications cause the most bottlenecks (example: virtual function calls in object oriented languages result in branches that are difficult to predict) can trigger research in architectures and compiler optimizations to handle the bottleneck. If desired, it can also guide software developers to tune their applications for performance.

Workload characterization can also be used to validate trace reduction and trace sampling techniques used in performance evaluation. Iyengar et. al. [Iy96] present a technique to reduce long traces to representative short traces. Characterization of intrinsic features of the trace before and after reduction can be employed to verify representativeness of the trace reduction technique. If representativeness is established based on isolated performance measurements, it is possible that two discrepancies are compensating each other. Characterizing distinct features can establish representativeness on a broad spectrum.

Synthetic benchmarks have traditionally been looked down, however, if processors of tomorrow have to be designed on applications other than programs of yesterday, it is important to be able to speculatively synthesize workloads. Identifying and characterizing the intrinsic properties of an application in terms of its memory access behavior, locality, control flow behavior, instruction level parallelism, etc. can lead to useful synthetic workloads modeling emerging or future applications. Poursepanj et. al. synthesized graphics workloads based on measurements on existing workloads and extrapolating anticipated features of future workloads [Po99]. Previous effort on workload synthesis include [Si82] [We90] [Wo88].

Eventually, workload characterization should lead to a program behavior model, which can be used in conjunction with a processor model to do analytical performance modeling of computer systems. The performance of a computer system for a workload is determined by the system itself (hardware and software) and the characteristics of the application. Majority of pre-silicon system performance evaluation research is performed using simulations. The workloads used in these simulations typically include programs

* This work is supported in part by the National Science Foundation under grants CCR-9697098 (CAREER Award), EIA-9807112, the State of Texas Advanced Technology Program, IBM, Dell, Intel, AMD and Microsoft.

from the SPEC95, SPECJVM, Winstone, or similar other benchmark suites. Many of these benchmarks have been constructed to be large to avoid entirely filling in the large caches found in modern processors. The large amounts of time needed for simulation experiments with these large benchmarks often prevent designers and architects from exploring a large span of the design space. Gee et. al. [Ge95] mention that cache simulations on SPEC92 benchmarks for varied cache parameters consumed 3 years of computing time. The simulations do not even yield information on how the particular cache affects processor performance; they only yielded cache miss ratios. If one could ever build reasonably accurate analytical models for evaluating computer performance, the models would have been fast enough to allow a thorough evaluation of vast regions of the design space. While analytical models may not be very accurate, preliminary design space exploration can be performed by them followed by simulation models for detailed evaluation of selected architectures.

Creating abstract models for computer performance analysis would necessitate creation of models for the system as well as the application (Figure 1). Analytical modeling of computers in the past has relied on techniques such as probabilistic models, queuing theory, Markov models, Petri nets, etc. [He84][Ja91][Kr96][Tr82]. Analytical models have been successfully used in the analysis of performance of large computer systems containing networks of several processors, multiprogrammed systems, multicomputer networks etc. However, evaluators of uniprocessors have primarily been using the quantitative approach. One reason for the lack of popularity of analytical models for processor performance analysis has been the inaccuracy of results. It is extremely difficult to create accurate analytical models of complex processor architectures. Difficulty in formulating program behavior models has led to the use of heavily generalized input models. Unless inputs to the system can be modeled and fed in properly, the outputs from the model would be far from accurate.

Recently there has been some concerns on the accuracy of simulation models [Bl96] and a renewed interest in analytical modeling of processors and systems [No97] [Sm98] [So98]. Creation of an adequate program behavior model would catalyze this research and facilitate the path to a scientific method for computer performance evaluation. While challenging, if successful, it can immensely help rapid exploration of the design space.

Simulation studies are mostly guided by popular benchmark suites, which may not span over the entire potential application space. It is quite possible that the current suite of benchmarks are clustered over certain regions of the input space as shown in Fig. 2. This may be adequate for architectures targeted to specific applications, but general purpose architectures are often targeted to wider input space.

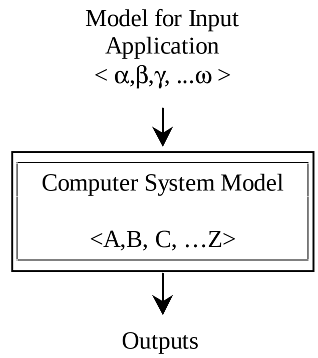


Figure 1. System Model

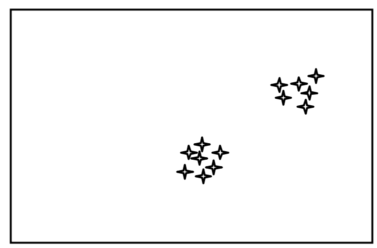


Figure 2 Potential span of existing benchmark suites in the potential workload space

For general purpose applications, at the onset of a new application (eg: web applications, mobile computing) or programming paradigm (eg: object oriented programming, Java), all simulations have to be repeated. Having a general program behavior model can allow experiments on applications with very different characteristics than what is available in popular benchmark suites. If benchmarks used in performance evaluation covered every region in the input space, as in Fig. 3, performance characterization using techniques such as trace-driven or execution driven simulations would have been sufficient although cumbersome. Assume that current benchmarks consist of *App1* and *App2*, developed in programming languages 1 and 2 as shown in Figure 4. A radically new application in an existing language or an existing application in a new programming paradigm may lead to inputs with widely different characteristics. For example, behavior of applications developed in Java is influenced predominantly by the Java Virtual Machine and the mode of execution (interpreter versus Just-in-Time compiler) rather than the intrinsic nature of the application itself. A general program behavior model can allow the creation of benchmarks that represent potential emerging or future applications or future programming languages, before the advent of the actual applications.

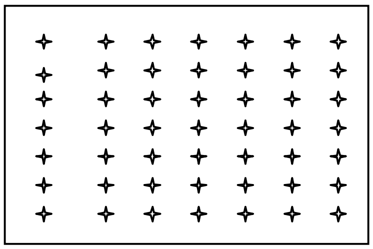


Figure 3. Potential application span

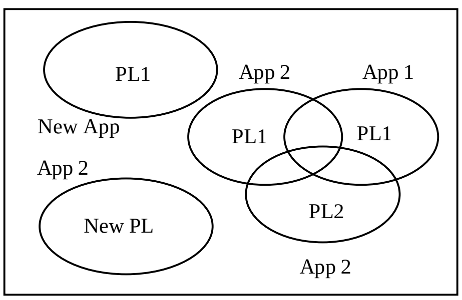


Figure 4: Changes in application domain with new languages and programming paradigms

In summary, workload characterization can serve the following purposes:

1. Understand workloads and interpret simulation results effectively
2. Design machines to suit particular workloads/tune micro-architecture to suit particular features in programs
3. Validate trace reduction mechanisms
4. Validate representativeness of sampled traces
5. Guide selection of programs for benchmark suites
6. Guide selection of a limited set of programs from benchmark suites, if there are time limitations to evaluate the entire suite
7. Generate synthetic traces to model emerging/future workloads
8. Create an abstract program behavior model, which in conjunction with a processor model can do analytical performance modeling of computer systems.

Several early studies on program behavior are available in literature [Ag76] [Ag77] [Al90] [Da87] [De68] [Du94] [Fe81] [Fr75] [Ha77] [Ha82] [Ha72] [Har82] [Ha83] [Sp72] [Za83]. Irrespective of the differences of modern workloads from the workloads used, the methodologies are worth learning from.

2. List of Workload Characteristics

Completely characterizing even a simple object can be challenging. Consider describing an "apple" in terms of its properties like shape, size, color, texture, etc. Several size, shape and surface primitives must be described in order to be able to clearly recognize the object as an "apple". Now, if one were to describe a program in terms of several intrinsic properties of programs, what would be the properties in that list? If $\alpha,\beta,\gamma, ...\omega$ were a list of properties that completely describe the characteristics of a program, a program could be represented as a vector $< \alpha,\beta,\gamma, ...\omega >$. A partial list of these properties is provided in this section.

Workload characteristics can be implementation (machine) dependent or independent. Ideally one would like to obtain generic or architecture independent properties of programs, in order to create a generic program behavior model. However, often it is nearly impossible to obtain completely implementation independent characteristics, as there are compiler, library and operating system influences. Effect of the target processor's instruction set architecture (ISA), its register file size, etc will also be reflected in the program characteristics, unless characterization is at source code level. Majority of the ongoing workload characterization effort is system level characterization and reflects machine properties also, rather than pure workload characterization.

2.1 Characteristics at source code level

Weicker's characterization of several stone age benchmarks [We90] includes characteristics such as statement distribution in programs (assignment of variables, constants, if-statements, FOR, Goto, While/repeat, Switch, break, Call, return, etc), distribution of operand data types (integer, character, float/double, boolean, array, string, pointer etc.) and distribution of operations (integer add/subtract/multiply/divide, floating point add/subtract/multiply/divide, relational, logical etc.). Conte et. al. [Co99] presented characteristics such as applet redundancy, method redundancy and method overlap for Java applications. Source level characteristics can yield overall information on the nature of the application, particularly the nature of the control flow, basic block sizes, instruction run lengths, code reuse, and predictability of branches. For example, abundance of IF-THEN-ELSE statements point to irregular and often unpredictable control flow while loops indicate structured control flow. The abundance of switch statements as in the Java Virtual Machine (in the interpreted mode) can point to indirect branches, whose targets are often difficult to predict.

2.2 Micro-architecture Independent Characteristics

While micro-architecture independent, these attributes are influenced by language, compilers, operating system and ISA. Some of these attributes are often characterized by a microarchitecture dependent metric, for example, cache hit ratios are used by many to describe program locality [Sa98]. Similarly, the profile of dependency distances in a program can be used as a measure of parallelism in a microarchitecture independent fashion, while many use instructions per cycle (IPC) to indicate available parallelism in a machine dependent fashion. Noonburg et. al. [No97] and Dubey et. al [Du94] describe some generic measures of parallelism, while Austin et. al. [Au92] construct a dependency graph, identify the critical path and compute available parallelism in a more architecture dependent fashion. The following is a partial list of commonly studied characteristics:

- Foot print/static size (in instructions and in bytes)
- Dynamic size (in instructions}
- Ratio of dynamic to static size
- Instruction mix
- Computation to memory access ratio (memory bound versus computation bound)
- True computations versus address computation [Ha77][Ku93][Jo95]
- Overhead memory accesses versus true memory accesses [Ha77][Ku93][Jo95]
- Computation to communication ratio
- Fraction of operating system code versus user code
- Nature of accesses (random versus structured)
- Basic block size
- Control transfer instruction mix (direct branches, indirect branches, calls, jumps, returns)
- Predictability of control structures
- Percent branches with multiple targets
- Consistency of branch targets
- Temporal locality of instructions
- Temporal locality of data
- Spatial locality of instructions
- Spatial locality of data
- Non-cacheable storage references
- Available parallelism
- Data parallelism
- Instruction level parallelism
- Granularity of parallelism (fine-grain versus coarse grain versus massive parallelism)
- Dependency profile
- Average dependency distance
- I/O accesses per transaction
- I/O request patterns (read/write ratio, sequential versus random)

2.3 Micro-architecture Dependent Characteristics

Majority of ongoing workload characterization is performed in a microarchitecture dependent manner, primarily because the objective is to study the match between applications and architectures [Ba98] [Bh97] [Ke98] [Le98] [Lo98] [Wo95]. The following is a partial list of commonly studied characteristics:

- Cache hit ratios
- Cache bus utilization/traffic ratio
- Cache replacement generated traffic
- Multiprocessor cache coherency related traffic
- System bus bandwidth consumption
- Branch prediction rate

- Available parallelism
- Throughput (operations or transactions/minute)
- Disk throughput
- Communication bandwidth
- Disk read fraction
- Prefetching effectiveness
- Time spent in operating system code versus user code
- Scalability of multiprocessing related characteristics
- Computation to communication ratio in multiprocessors
- Concurrency and load balance in multiprocessors
- Inter-node communication rates and message sizes

3. A Taxonomy for Program behavior

Current classifications of programs include generalizations like memory intensive, compute intensive, operating system intensive, etc. Once a large list of program properties {α,β,γ, ...ω} are determined, we could classify programs in the following manner.

Type A programs – {α=a,β=aa,γ=aaa, ...ω=aaaa}

Type B programs – {α=b,β=bb,γ=bbb, ...ω=bbbb}

Type C programs – {α=c,β=cc,γ=ccc, ...ω=cccc}

Type D programs - {α=d,β=dd,γ=ddd, ...ω=dddd}

Type X programs - {α=x,β=xx,γ=xxx, ...ω=xxxx}

However, considerable effort is required to identify all relevant parameters and to develop appropriate metrics to represent each, before such a classification can be performed. The performance of an application on a system can be obtained by combining the system model with the program model as was illustrated in Figure 1.

4. Characterization Methods

Workload characterization can be performed using hardware and software methodologies. **Hardware** measurements typically involve machine (implementation) dependent characteristics. **Off/chip** hardware instrumentation such as logic analyzers [Po98], microcoded instrumentation [Ag86], or **on-chip** performance monitoring counters [De97] [Bh97] [Lu99] [Ra99] are typically used. **Software** instrumentation can be used for implementation dependent/independent measurements. **Profiling** [Ka99] or **simulation** [Gr99] [Br99] [Ki99] techniques are typically used. Vtune [Bh98], Shade [Co95], pixie [Co95], ATUM [Ag86] etc are popular tools for the x86, SPARC, MIPS and Alpha architectures respectively. Conte and Gimarc [Co95] provide details on several profiling and instrumentation tools.

5. Example Generic Metrics – Intrinsic Metrics of Locality

Locality of reference has always been an interesting program property, because it has a fundamental impact on memory hierarchy design and system performance. Two or three decades ago, researchers on virtual memory [De68][Sp72] developed several program locality models, for example, working set models, LRU stack models, Independent Reference Models (IRM) etc. At the same time, in most recent research, locality is characterized in terms of cache hit ratios [Sa98]. No generic or abstract or intrinsic characterization of locality on any modern benchmark suite has been attempted. Ideally it should be possible to arrive at a locality index or locality function for every application. This locality function should be useful in predicting cache performance without detailed simulations. Conversely it should be possible to generate workloads with arbitrary amounts of locality and use them for computer system performance evaluation, once appropriate metrics of locality are identified.

Temporal locality is often defined as follows: "If an item is referenced, it will tend to be referenced again soon". Spatial locality is commonly defined as follows: "If an item is referenced, items whose addresses are close by will tend to be referenced soon". Conte and Hwu [Co90] presented some excellent measures of locality. Two of the measures described in their research are the *inter-reference temporal density function* and the *inter-reference spatial density function.*

The *inter-reference temporal density* function, $f^T(x)$ is defined as the probability of there being x unique references between successive references to the same item.

Example

Consider the following sequence of instruction addresses:

3000, 3002, 3004, 3000, 3004, 3000, 3002

The number of unique address references x between the first reference to address 3000 and the next reference to it is 2. The number of unique address references between the first reference to address 3002 and the next reference to it is also 2 (there are 4 references but only 2 unique references). Similarly, there is only one unique reference between the two consecutive references to address 3004 and between the second and third reference to address 3000. Thus if we consider the instruction reference stream as a whole, there are two instances where x has value 1 and two instances where x has value 2. We normalize these counts over all possible values of x to get the inter-reference temporal density function $f^T(x)$ as $f^T(0) = 0$, $f^T(1) = 0.5$ $f^T(2) = 0.5$.

The *inter-reference spatial density function* $f^S(x)$ is defined as the probability that between references to the same item, a reference to an item x units away occurs.

Example

Consider the following address reference stream:

3000, 3004, 3008, 300C, 3004, 3000

There are 2 references to an item that is 4 units away (i.e. 3004) between the first and second references to address 3000. Similarly, there is a reference to an item that is 4 units away (i.e. 3008) between the first and second references to address 3004. Thus there are 3 instances where there is a reference to an item that is 4 units away between two consecutive references to an item, 2 instances where there is a reference to an item that is 2 units away and 1 instance where there is a reference to an item that is 12 units away. Normalizing gives the inter-reference spatial density function for different values of x as $f^S(4) = 0.5$, $f^S(8) = 0.3333$ and $f^S(12) = 0.167$.

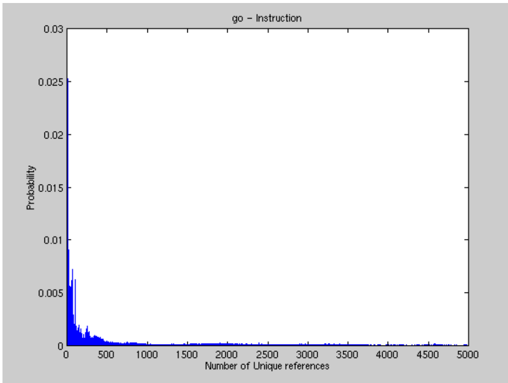
Conte and Hwu evaluated these locality density functions for applications such as *grep* and *yacc* [Co90]. We evaluated these temporal and spatial locality measures for the SPEC95 integer programs. Prior research presents only probability density functions, whereas the probability distribution functions (pdf) can be integrated to obtain the cumulative distribution function (cdf), which provides a good indication of the locality. As an example, we illustrate the probability density function and the cumulative distribution functions of three programs, *go*, *li* and *mk88sim*, in Figure 5 and Figure 6 respectively. The spatial locality plots have address distance on the x-axis and the temporal locality plots have number of unique references on the x-axis. Strong locality will be displayed as lower rise times or higher gradient (slope) in the cumulative distribution function. Table 1 illustrates that the temporal locality of *go* is poor while that of *li* is strong. Similarly, it may be observed from Figure 6 that the spatial locality of *go* is also extremely poor. The 20 percentile point is reached in 284 unique references in *li*, while it takes 2000 unique references to reach the equivalent point in *go*. Measurement results using performance counters on an UltraSPARC [Ra98] shows miss rates of 5.66%, 0.36% and 2.82% for *go*, *li* and *mk88sim* respectively, which correlates well with the observed locality profiles. (The UltraSPARC has a 16 kbyte instruction cache.)

No of uniques	go	li	m88ksim
256	0.45	0.69	0.67
512	0.59	0.88	0.70
1024	0.68	0.99	0.94
2048	0.79	0.99	0.98

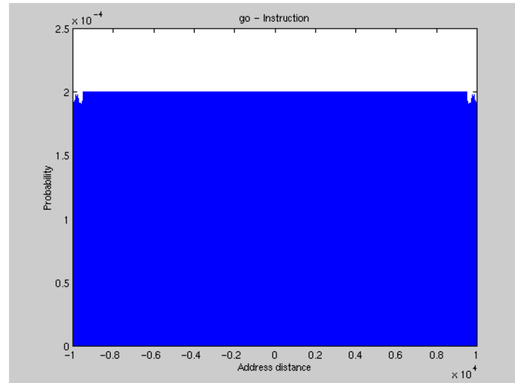
Table 1. Summary of the temporal locality charts. Cumulative percentile values corresponding to specified number of unique references in between two successive references to the same item.

Locality characterizations can be extremely useful in tuning cache performance. The distinct characterization of temporal and spatial locality aspects can assist to tune schemes such as dual caches [Go95] which provide separate mechanisms to capture temporal and spatial locality. Instead of splitting the cache area equally between temporal and spatial caches, dynamic partitioning depending on strength/weakness of each kind of locality can be employed.

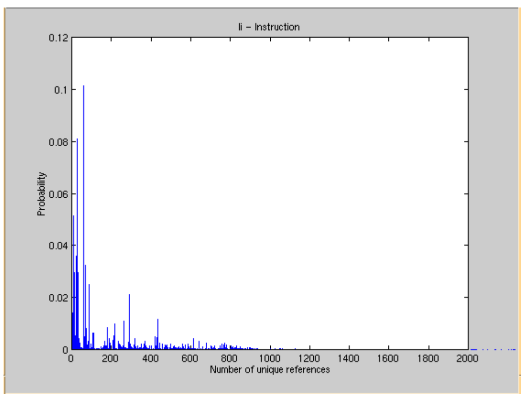
The locality metrics illustrated in this section are by no means total indicators of program locality. One inadequacy of the temporal locality metric presented here is that the frequency of usage is not reflected in this measure. This metric cannot distinguish between $(abc)^{10}$, $(abc)^{100}$ and $(abc)^{1000}$. Temporal locality of all these sequences would evaluate to $f^T(2)= 1$, which is not desirable. If all aspects of locality can be captured into the locality profiles, it should be possible to create an accurate cache model that derives cache hit ratios given cache parameters and the locality profiles. Analytical cache modeling has been attempted in the past by several researchers including Agarwal et. al. [Ag89], Ghosh et. al. [Gh97], Thiebaut et. al. [Th87],



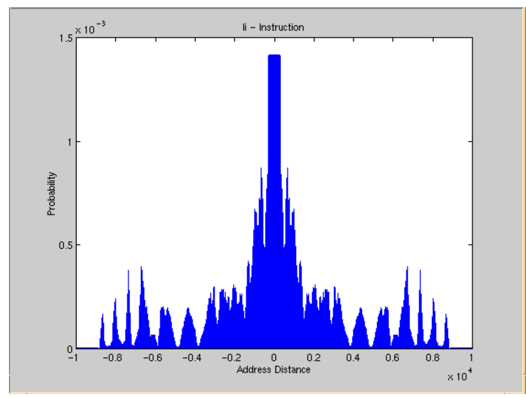
Temporal locality for *go*



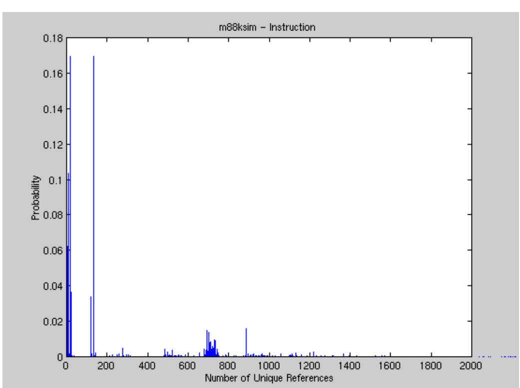
Spatial locality for *go*



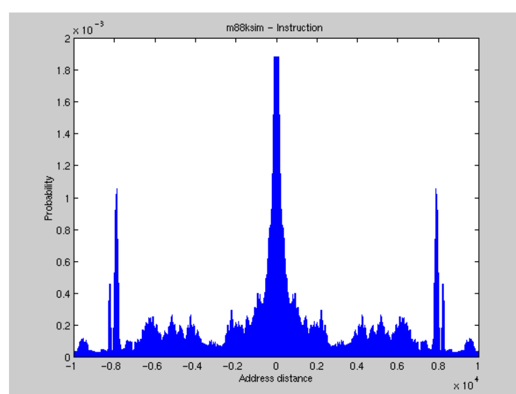
Temporal locality for *li*



Spatial locality for *li*

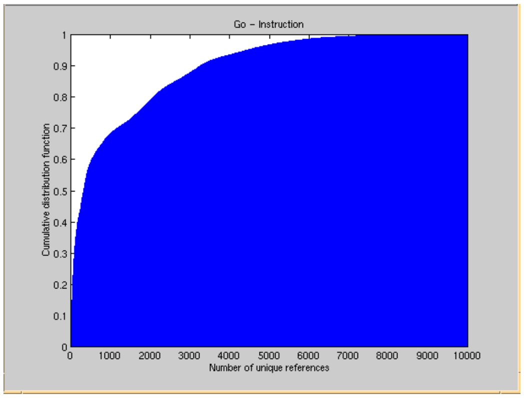


Temporal locality for *m88ksim*

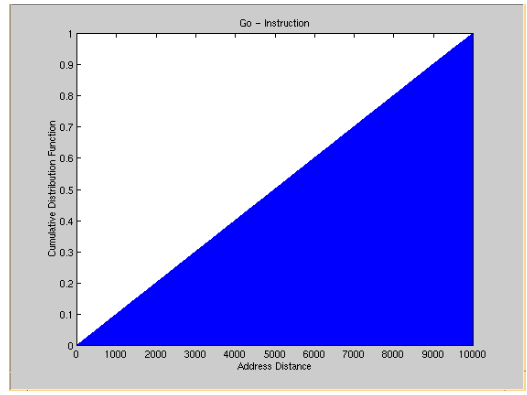


Spatial locality for *m88ksim*

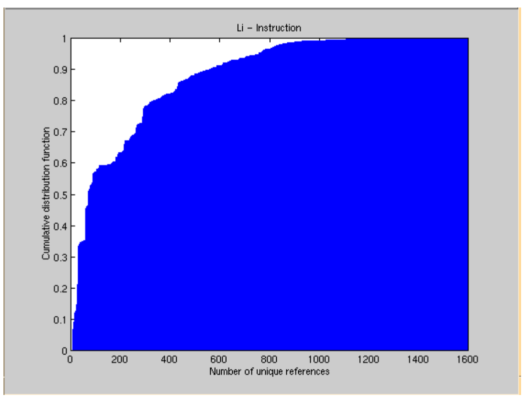
Figure 5 Probability density functions for temporal and spatial locality



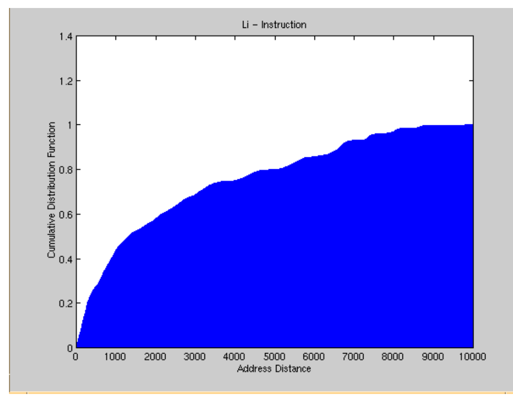
Cumulative distribution for *go* - temporal



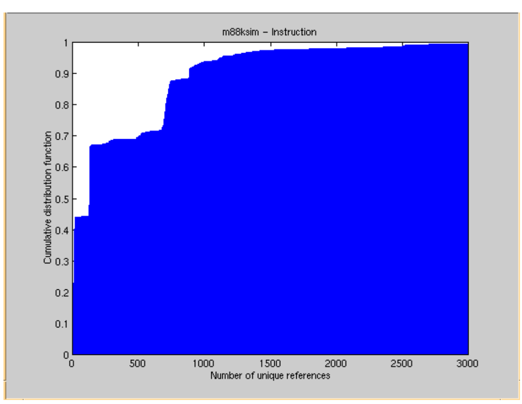
Cumulative distribution for *go* - spatial



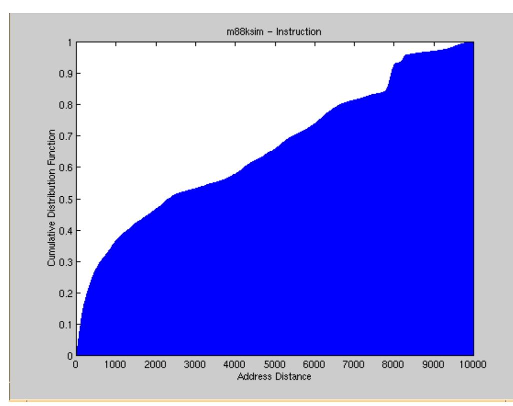
Cumulative distribution for *li* - temporal



Cumulative distribution for *li* - spatial



Cumulative distribution for *m88ksim* - temporal



Cumulative distribution for *m88ksim* - spatial

Figure 6 Cumulative distribution functions for temporal and spatial locality

Mendelson, et. al. [Me93], and Mowry et. al. [Mo97]. In order to correlate well with cache replacement policies and mapping schemes, all aspects of locality need to be reflected in locality metrics.

6. Concluding Remarks

In this paper, we presented a set of goals that can be realized using workload characterization. In the short term, workload characterization is expected to impact performance tuning of architectures for emerging workloads. It can also guide tuning of compiler optimizations and application development. The long term objectives for workload characterization consist of the creation of a program behavior model, synthesis of future workloads, and analytical performance modeling and design space exploration. The ability to analyze applications with fundamental measures of behavior can boost the development of analytical models to evaluate performance.

References and Further Reading

[Ag89] A. Agarwal, M. Horowitz, and J. Hennessy, "An analytical cache model", ACM Transactions on Computer Systems, vol. 7, pp. 184-215, May 1989

[Ag76] A. K. Agrawala, R. M. Bryant, and J. M. Mohr, "An approach to the workload characterization problem", Computer, vol. 9, pp. 18-32, 1976

[Ag77] A. K. Agrawala and J. M. Mohr, "The relationship between the pattern recognition problem and the workload characterization problem", Proceedings of SIGMETRICS/CMG VIII, 1977, pp. 131-139

[Ag86] A. Agarwal, R. L. Sites and M. Horowitz, "ATUM: A New Technique for Capturing Address Traces Using Microcode"" Proceedings of the 13th International Symposium on Computer Architecture, June 1986, pp. 119-127.

[Al90] T. Alanko, I. Haikala and P. Kutvonen, "Methodology and Empirical Results of Program Behavior Measurements", Proceedings of the 7th IFIP W.G.7.3 International Symposium on Computer Performance, Modeling, Measurement and Evaluation, Toronto, May 1990, pp. 55-66

[Au92] T. M. Austin and G. S. Sohi, "Dynamic Dependency Analysis of Ordinary Programs", Proceedings of the International Symposium on Computer Architecture, 1992, pp. 342-351.

[Ba98] L. Barroso, K. Ghorachorloo and E. Bugnion, "Memory System Characterization of Commercial Workloads", Proceedings of the International Symposium on Computer Architecture, 1998, pp. 3-14

[Bh97] D. Bhandarkar and J. Ding, "Performance Characterization of the PentiumPro Processor", Proceedings of the 3rd High Performance Computer Architecture Symposium, 1997, pp. 288-297.

[Bh98] R. Bhargava, L. K. John, B. L. Evans, R.Radhakrishnan, Evaluating MMX Technology Using DSP and Multimedia Applications, Proceedings of the IEEE Symposium on Microarchitecture(MICRO-31), Dallas, Texas, pp.37-46, Dec.1998.

[Bl96] B. Black, A. S. Huang, M. H. Lipasti and J. P. Shen, " Can trace-driven simulator accurately predict superscalar performance", Proceedings of International Conference on Computer design (ICCD) 1996, pp. 478-485.

[Br99] J. P. Bradford, and J. Fortes, "Performance and Memory-Access Characterization of Data Mining Applications", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE CS Press, 1999

[Co90] T. Conte and W-m W. Hwu, "Benchmark Characterization for Experimental System Evaluation," Proceedings of the 1990 Hawaii International Conference on System Sciences (HICSS), Vol. I, Architecture Track, pp. 6-18

[Co95] T. M. Conte and C. E. Gimarc, "Fast Simulation of Computer Architectures", Kluwer Academic Publishers, 1995.

[Co99] M. T. Conte, A. R. Trick, J. C. Gyllenhaal, and W-m W. Hwu, "A Study of Code Reuse and Sharing Characteristics of Java Applications", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE CS Press, 1999

[Da87] F. Darema-Rogers, G. F. Pfister, and K. So, "Memory Access Patterns of Parallel Scientific Programs", Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Banff, Canada, May 1987, pp. 46-58.

[De97] J. Dean, J. E. Hicks, C. A. Waldspurger, W. E. Weihl, and G. Chrysos, "ProfileMe: Hardware Support for Instruction-Level Profiling on Out-of-order Processors", Proceedings of the 30th International Symposium on Microarchitecture (Micro-30), December 1-3, 1997, pp. 292-302.

[De68] P. J. Denning, "The Working Set Model for Program Behavior", Communications of the ACM, Volume II, Number 5, May 1968, pp. 323-333.

[Du94] P. K. Dubey, G. B. Adams, and M. J. Flynn, Instruction Window Size Trade-Offs and Characterization of Program Parallelism, IEEE Transactions on Computers, Vol. 43, No. 4, April 1994, pp. 431-442

[Em86] P. G. Emma and E. S. Davidson, "Characterization of branch and data dependencies in programs for evaluating pipeline performance", RC 11733 52535, Computer Science Dept, IBM T. J. Watson Research Center, Yorktown Heights, NY 1986.

[Ev98] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt, "An Analysis of Correlation and Predictability: What makes Two-Level Branch Predictors Work", Proceedings of the International Symposium on Computer Architecture, 1998, pp. 52-61.

[Fe81] D. Ferrari, "A performance oriented procedure for modeling interactive workloads", in Experimental Computer Performance Evaluation, D. Ferrari and M. Spadoni, Eds, Amsterdam, The Netherlands: North Holland, 1981, pp. 57-78.

[Fr75] W. F. Freiberger, U. Grenander, and P. D. Sampson, "Pattern in Program References", IBM Journal of Research and Development, May 1975, pp. 230-243.

[Ge95] J. D. Gee, M. D. Hill, D. N. Pnevmatikatos, A. J. Smith, Cache Performance of the SPEC Benchmark Suite. Technical Report #1049. University of Wisconsin Madison.

[Gh97] S. Ghosh, M. Martonosi, and S. Malik, "Cache Miss Equations: An Analytical representation of Cache Misses", Proceedings of the Workshop on Interaction between Compilers and Computer Architectures, Feb 1997, San Antonio, Texas.

[Go95] A. Gonzalez, C. Aliagas and M. Valero, "A data cache with multiple caching strategies tuned to different types of locality", Proceedings of the International Conference on Supercomputing, 1995, pp. 338-347.

[Gr99] B. Grayson, and C. Chase, "Characterizing Instruction Latency for Speculative Issue SMPs: A Case Study of Varying Memory System Performance on the SPLASH-2 Benchmarks", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE CS Press, 1999

[Ha77] Hammerstrom, Davidson, "Information Content of CPU memory referencing behavior", 4th Annual Symposium on Computer Architecture, March 1977, pp. 184-192.

[Ha82] R. B. Hagman and R.S. Fabry, "Program Page Reference Patterns", Proceedings of the 1982 ACM Sigmetrics Conference on Measurements and Modeling of Computer Systems, Seattle, WA, 1982, pp. 20-29.

[Ha72] D. J. Hatfield, "Experiments on Page Size, Program Access Patterns and Virtual Memory Performance", IBM Journal of R & D, Vol. 16, No. 1, Jan 1972, pp. 58-66.

[Har82] G. Haring, " On state-dependent workload characterization by software resources", in Performance Evaluation Review, 11, Proceedings of 1982 ACM SIGMETRICS Conf. Meas. Modeling Computer Systems, pp. 51-57.

[Ha83] G. Haring, "On stochastic models of interactive workloads", in PERFORMANCE '83. A. K. Agrawala and S. K. Tripathi, Eds, Amsterdam, The Netherlands: North-Holland, 1983, pp. 133-152.

[He84] P. Heidelberger and S. S. Lavenberg, Computer Performance Evaluation Methodology, IEEE Transactions on Computers, Dec 1984, pp. 1195-1220

[Iy96] V. S. Iyengar, L. H. Trevillyan, P. Bose, Representative Traces for Processor Models with Infinite Cache, Proceedings of HPCA-2, 1996.

[Ja91] Raj Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling," Wiley- Interscience, New York, NY, April 1991

[Jo95] L. K. John, V. Reddy, P. Hulina, and L. Coraor, "Program Balance and its Impact on High Performance RISC Architectures, Proceedings of the International

Symposium on High Performance Computer Architecture, pp.370-379, Jan. 1995.

[Ka99] J. Kalamatianos, R. Chaiken, and D. Kaeli, "Parameter Value Characterization of Windows NT-based Applications", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE CS Press, 1999

[Ke98] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael and W. E. Baker, "Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads", Proceedings of the International Symposium on Computer Architecture, 1998, pp. 15-26

[Ki99] J-S Kim, X. Qin, and Y. Hsu, "Memory Characterization of a Parallel Data Mining Workload", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE CS Press, 1999

[Kr96] C. M. Krishna, "Performance Modeling for Computer Architects, " IEEE Computer Society Press

[Ku93] L. Kurian (John) "Issues in the Design of a Decoupled Architecture for a RISC Environment", Ph. D dissertation, The Pennsylvania State University, August 1993.

[Le98] D. Lee, P. Crowley, J. Baer, T. Anderson and B. E. Bershad, "Execution Characteristics of Desktop Applications on Windows NT", Proceedings of the International Symposium on Computer Architecture, 1998, pp. 27-38.

[Lo98] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy and S. S. Parekh, "An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors", Proceedings of the International Symposium on Computer Architecture, 1998, pp. 39-50.

[Lu99] Yong Luo, and Kirk W. Cameron, "Instruction-level Characterization of Scientific Computing Applications using Hardware Performance Counters", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE CS Press, 1999

[Me93] A. Mendelson, D. Thiebaut and D. K. Pradhan, "Modeling Live and Dead Lines in Cache Memory Systems", IEEE Transactions on Computers, Vol. 42, No. 1, Jan 1993.

[Mo97] T. C. Mowry and C-K Luk, "Predicting Data Cache Misses in Non-Numeric Applications Through Correlation Profiling", Proceedings of the 30th International Symposium on Microarchitecture (Micro-30), December 1-3, 1997, pp. 314-320.

[No97] D. B. Noonburg and J. P. Shen, A Framework for Statistical Modeling of Superscalar Processor Performance, Proceedings of the 3rd International Symposium on High Performance Computer Architecture (HPCA), 1997pp. 298-309.

[Po99] Ali Poursepanj and David Christie, "Generation of 3D Graphics Workload for System Performance Analysis", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE CS Press, 1999

[Ra98] R. Radhakrishnan and L. John, "Execution Characteristics of Object Oriented Programs on the UltraSPARC-II, Proceedings of the 5th International Conference on High Performance Computing, Dec. 1998.

[Ra99] R. Radhakrishnan, and F. L. Rawson, "Characterizing the Behavior of Windows NT Web Server Workloads using Processor Performance Counters", Presented at the First Workshop on Workload Characterization, Also in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE CS Press, 1999

[Sa98] F. J. Sanchez and A. Gonzalez, Data Locality Analysis of the SPECfp95, Digest of Performance Analysis and its Impact on design (PAID) Workshop conducted in conjunction with ISCA-98, Spain, Barcelona. pp. 78-84.

[Si82] A. Singh and Z. Segall, "Synthetic Workload generation for experimenting with multiprocessors", Proceedings of the 3rd International Conference on Distributed Computing Systems, 1982, pp. 778-785.

[Sm98] Transparencies from Jim Smith's talk on Modeling Superscalar Processors via Statistical Simulation, authored by Richard Carl and Jim Smith, PAID Workshop, Barcelona, Spain, 1998.

[Sp72] J. R. Spirn and P. J. Denning, "Experiments with Program Locality", The Fall Joint Computer Conference, 1972, pp. 611-621.

[So98] D. J. Sorin, V. S. Pai, S. V. Adve, M. K. Vernon, D. A. Wood, “Analytic Evaluation of Shared Memory Systems with ILP Processors”, Proceedings of the 25th Annual International Symposium on Computer Architecture, June 1998, pp. 380-391

[Sr74] K. Sreenivasan and A. J. Kleinman, “On the construction of a representative synthetic workload”, Communications of the ACM, vol. 17, -p. 127-133, 1974.

[Th87] D. Thiebaut and H. S. Stone, “Footprints in the Cache”, ACM Transactions on Computer Systems, Vol. 5, No. 4, Nov. 1987, pp. 305-329.

[Tr82] K. S. Trivedi, Probability and Statistics with Reliability, Queueing and Computer Science Applications”, Englewood Cliff: Prentice Hall, 1982

[We90] R. P. Weicker, “An Overview of Common Benchmarks”, IEEE Computer magazine, December 1990, pp. 65-75

[Wo88] W. S. Wong and R. J. T. Morris, Benchmark Synthesis Using the LRU Cache Hit Function, IEEE Transactions on Computers, Vol. 37, No. 6, June 1988, pp. 637-645.

[Wo95] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A.Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations”, Proceedings of the 22nd International Symposium on Computer Architecture, pages 24-36, June 1995.

[Za83] J. Zahorjan, “ Workload representation in queueing models of computer systems”, in Performance Evaluation Review Special Issue, Proc 1983 ACM Sigmetrics Conf measurement and Modeling of Computer Systems, pp. 70-81.