U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NISTIR 4418

National Computer Systems Laboratory

State Occupancy Information for Performance Comparisons

Gordon E. Lyon

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
National Computer Systems Laboratory
Advanced Systems Division
Gaithersburg, MD 20899

October 1990

CMRF

COMPUTER MEASUREMENT
RESEARCH FACILITY
FOR HIGH PERFORMANCE
PARALLEL COMPUTATION

Partially sponsored by the Defense Advanced Research Projects Agency

State Occupancy Information for Performance Comparisons

G.E. Lyon

Advanced Systems Division
National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

• Sponsored in part by the Defense Advanced Research Projects Agency.

Text of 12 October 1990.

U.S. Department of Commerce, Robert A. Mosbacher, Secretary

National Institute of Standards and Technology
John W. Lyons, Director

October 1990

TABLE OF CONTENTS

Page

Performance Characterization via State Occupancies 2

Comparison with Related Work 2

Other Representations 3

States and Macrostates 3

Local State Information 6
Measuring or Estimating 7

Mean Node Occupancies and Three Hypercube Paradigms 7

Discussion and Results 8

Problems in the Data 9
Estimators and Modeling 9
On a Real System 10

Local State Information from Amorphous Systems 10

Summary and Conclusion 11

Conclusion 12

Citations 12

State Occupancy Information for Performance Comparisons

G.E. Lyon

A state-based performance characterization attaches fixed processing rates to each service state. However, the number of states can be large. Over time, the sequences of such states are enormous. Counts of active (but interchangeable) system elements define macrostates, which are fewer. Furthermore, only the occupancy levels of macrostates are recorded. This removes time sequencings as a combinatoric problem, but still captures general performance details. Applications can be compared independently of their algorithmic structures.

The hypercube and other distributed-memory systems bring both opportunities and challenges to a state-based approach. Certainly, processor and communication activities are more easily identified and more independent with distributed-memory than with shared-memory. But isolated nodes also entail problems in capturing global observation states. However, hypercube application codes are commonly homogeneous across nodes, so that aggregating local state information works well. Three paradigms illustrate homogeneous applications with communication dependencies that are strong (global), moderate (local), or weak (independent once spawned). The three performance summaries are accurate and extremely compact.

Key words: application comparisons; distributed-memory; performance; system states.

Users want accurately to compare general performances of applications on newer computer architectures. They want this without the tremendous complications of algorithmic structures or explicit time. Benefits of such an approach are clear: users save time and reduce risks of bad program-architecture matching; they build insight on the system needs of classes of programs; their overall picture of computations is simpler. Certainly, a simplified perspective of applications on a system encourages comparisons among them. Since all programs on a system use its common and available resources, a characterization at the system-resource level can support application comparisons.

The discussion assumes that collection accuracy is not a problem, even though on parallel processing systems, measurement perturbation can cause severe distortions in performance behavior. Measurements cause insignificant perturbation on the example system, which has custom hardware instrumentation [MCNR90]. The results indicate that in many circumstances the burden of instrumentation may be surprisingly light.

 Partially sponsored by the Defense Advanced Research Projects Agency, ARPA Task No. 7066.

Performance Characterization via State Occupancies

The programmer doing algorithm measurements and comparisons is constantly reminded that there are many distinct algorithms. And, while computer systems are potentially as varied as the programmed applications that run on them, few computer designs are actually built. This fact serves nicely to limit discourse. Keeping a focus upon real systems, the following approach is used:

1. Decompose each system into major service states that determine general performance. A given class of machines is treated as having but a limited set of dominant states, each state denoting a *fixed* set of processing capacities (rates).
2. View each application as a set of demands upon system service capacities (rates), with a *specific application demand* signified by an accumulative occupancy *in the corresponding system state.*
3. Build and interpret models based upon observed state occupancies and the associated rates.

The idea of observing service states enjoys simple but powerful advantages. The entire state space of responses is described. This space is closed, rather than open, and thus checks the resolution and consistency of measurements. Changes to input lead only to a redistribution of occupancies among service states. There arise *no* completely new behaviors. As a result, state-based measurement has structure. The fineness and number of the observation states hinge upon the resolution needed to explain system performances as input varies. Specific challenges include the need to ensure that states are succinctly defined and recorded, to manage a large number of states as parallel systems scale up in number of processors, and to compress state representations as systems do scale in size. Averages from local state information fall in the last category, and are discussed later for homogeneous codes on a (homogeneous) hypercube.

Comparison with Related Work

Recent workshop discussions with R. Saavedra-Barrara (U. CA.) and E. Miya (NASA) highlight differences in methodology (see table, below) between their efforts and the above approach [LYO90, SSM89]. First, they initiate application characterization at the language level--FORTRAN; their model (now) has 113 specific parameters, which reduce to perhaps 14 or more general factors [SSM89]. Their virtual FORTRAN machine model invites problems in performance variance acquired through layers of construction [LS90]. The model's level of abstraction is perhaps too high. In contrast, service states at a system-level characterization may generate only 4-10 parameters. (Later examples use four.) Hardware developed at NIST captures service state occupancies for our experiments; our group's advantage in making specialized instrumentation is pronounced. However, many instrumentations make only modest demands, such as a fast clock and several timer registers on each processor node. Secondly, the Berkeley-NASA researchers choose their dependent variables principally as statistical predictors on mostly serial machines, whereas our NIST observables are tied to true system states (actually, macrostates) and *are 100% measured.* On parallel architectures this difference can be very significant. For example, program communication latencies are much easier to measure (NIST approach) than to predict (Berkeley-NASA). Also, a state-based model has a formal structure that can be manipulated to advantage [LYO89]. Both approaches rely upon accumulative effects (e.g., time measured

overall) to remove the combinatorics of sequences (e.g., states in time).

Category	Approach	
	Berkeley-NASA [SSM89]	NIST [LYO89, LS90]
Objective	*compare and contrast architectures via standard FORTRAN virtual machine (includes applications)*	*compare applications concisely via major service states appropriate for each architectural family*
Application Treatment	*static: analyze source*	*dynamic: instrument, execute*
Host Treatment	*run "kernel" benchmarks to establish parameters*	*special measurement hardware captures information*
Modeling Effort	*complex: includes compiler actions, system behavior*	*simple* **if** *have well-chosen state variables*
Everyday Use	*easy, automatic*	*easy: system service statistics harder: specialized application data*
Added Hardware	*none*	*custom instrumentation*
Support Software	*customized for each system component (compiler, scheduler,...)*	*standard packages*
Philosophy Bias	*analytical*	*empirical*

Contrast in Approaches

Other Representations. States present another issue, that of practical representation. State tables shown in sequel are such that every attribute (column) appears in every state (row). However, when a column has little effect on most states (rows), it is convenient to select all unaffected rows, and to project a new subtable that eliminates the column. Repeated applications eventually yield many small subtables, none of which is complete in itself. Under these circumstances, a dependency tree works considerably better--it succinctly displays isolated clusters of interaction that leave other states essentially unaffected [LYO89, LS90]. However, in the context to follow, a table format is adequate.

States and Macrostates

Discussion of states and their role proceeds via a simple example. This should cause no loss of generality. Imagine a parallel system ABCD with four (p =4) processors: *a, b, c, and d*. The architecture for ABCD remains for the moment unspecified. At any time, each processor is attributed to one of three (m =3) major modes, A_1, A_2, or A_3. A concise description of system ABCD at time t_i is given as the *microstate*

	A_1	A_2	A_3	*Rate*
t_i:	a	b c	d	r_*(a, bc, d)

Microstate signifies that the detail of individual system elements is recorded. Rate $r_*()$ provides values for processor a in A_1, b and c in A_2, and d in A_3. This fully characterizes the performance of ABCD at time t_i. Table T1 shows a *time trace* of ABCD microstates. Time is defined by fields *index*, for order of microstate

occurrence, and *Duration*, for length of stay. The symbol "∅" denotes the empty set.

index	A_1	A_2	A_3	Duration	Rate
1	a b c d	∅	∅	3	r_*(abcd, ∅, ∅)
2	a c d	b	∅	1	r_*(acd, b, ∅)
3	a b d	c	∅	2	r_*(abd, c, ∅)
4	a c d	∅	b	1	r_*(acd, ∅, b)
5	a b c d	∅	∅	1	r_*(abcd, ∅, ∅)
6	a b	c d	∅	5	r_*(ab, cd, ∅)
7	c d	a b	∅	5	r_*(cd, ab, ∅)
8	a c	b d	∅	5	r_*(ac, bd, ∅)
9	∅	a b c d	∅	1	r_*(∅, abcd, ∅)
10	a b c d	∅	∅	2	r_*(abcd, ∅, ∅)
11	a	b c d	∅	2	r_*(a, bcd, ∅)
12	∅	abc	d	1	r_*(∅, abc, d)

T1: System ABCD Time Trace, Microstates

Microstates often contain much unnecessary detail. If ABCD is a good general-purpose design, all processing nodes likely have the same interconnect, the same processor chip, and the same memory architecture. (It simplifies manufacture, programming and maintenance.) Since no component is distinguished, *counts* of processors in a mode are what matter. Thus, the earlier microstate example

	A_1	A_2	A_3	*Rate*
t_i:	a	b c	d	r_*(a, bc, d)

becomes the *macrostate*

t_i:	1	2	1	r(1, 2, 1)

Table T2 shows that converting T1 to macrostates produces some adjacent rows that are identical except for index and duration. (Rows 2-3 and 6-8.) These groupings can be merged, adding their durations together. Rate r now takes integer arguments, rather than the sets of r_*.

index	A_1	A_2	A_3	Duration	Rate
1	4	0	0	3	r(4, 0, 0)
2, 3	3	1	0	3	r(3, 1, 0)
4	3	0	1	1	r(3, 0, 1)
5	4	0	0	1	r(4, 0, 0)
6, 7, 8	2	2	0	15	r(2, 2, 0)
9	0	4	0	1	r(0, 4, 0)
10	4	0	0	2	r(4, 0, 0)
11	1	3	0	2	r(1, 3, 0)
12	0	3	1	1	r(0, 3, 1)

T2: System ABCD Time Trace, Macrostates

The next simplification introduces *macrostate occupancy*, in contrast to sequences of macrostates *in time*. Selecting all T2 rows, project T2 on all columns except *index*. Excluding *Duration*, entries in the new table (T3) with duplicate fields merge into single rows, their *Duration* fields summing for a new *Occupancy* field. Now, while the set of macrostate sequences is infinite, macrostates for ABCD with $p=4$, $m=3$ are [FEL64]:

$$\binom{p+m-1}{p} = \binom{6}{4} = 15$$

The transformation to occupancies removes explicit time and the complications of algorithmic sequencings from the performance characterization. Useful remaining aspects of the application code include its independent variables and their settings during performance experiments. The result is table T3. (T3 does not show settings of the independent variables that generated it.)

A_1	A_2	A_3	Occupancy	Rate
4	0	0	6	r(4, 0, 0)
3	1	0	3	r(3, 1, 0)
3	0	1	1	r(3, 0, 1)
2	2	0	15	r(2, 2, 0)
0	4	0	1	r(0, 4, 0)
1	3	0	2	r(1, 3, 0)
0	3	1	1	r(0, 3, 1)

T3: System ABCD Occupancies, Macrostates of $A_1A_2A_3$

In T4 (below), all rows of T3 have been selected. A projection on A_1 also sums the occupancies of duplicate rows. As an example, T3 entries

A_1	A_2	A_3	Occupancy	Rate
0	4	0	1	r(0, 4, 0)
0	3	1	1	r(0, 3, 1)

become an entry

A_1	Occupancy	Rate
0	2	$r_1(0)$

in table T4. The transformation assumes either A_2 and A_3 are inconsequential to gross system performance and are ignored, or that their information is implicit in values taken by $r_1()$. A_1 is commonly level-of-parallelism for many parallel system analyses.

A_1	Occupancy	Rate
4	6	$r_1(4)$
3	4	$r_1(3)$
2	15	$r_1(2)$
1	2	$r_1(1)$
0	2	$r_1(0)$

T4: System Occupancies, Macrostates of A_1

Further compressions of state information are often necessary. For despite simplifications already introduced, the number of macrostates (expressed by the binomial coefficient) grows swiftly as systems scale up in processors. Often, a continuous function can be fit to give a compact approximation to a given attribute (column entries). The best-known example is probably the Hockney-Jesshope vector rate of $r_j(n) = r_\infty \left[1 + \frac{n_{½}}{n}\right]^{-1}$ associated with attribute $A_j \equiv$ *vector length*, n. As mentioned earlier, the practical representation of states is a somewhat distinct topic. See [LS90] for examples.

Local State Information. A more serious problem arises when immediate global knowledge of processors or other node activity is lacking (each column proclaims a known global level of activity for its attribute). Precise global timing is often not available on distributed systems. (Our NIST-instrumented iPSC-1 does global timings.) An obvious approach is to examine homogeneous collections of nodes--those identical in hardware and in assigned software tasks. This is not unrealistic, for most applications on hypercubes and related architectures are regular problem domains attacked via code replicated across nodes [FJL88]. The regularity of hypercube architecture invites such solutions. Observations made independently at each node are averaged together. Overall application consumption of system time is recorded at detail levels appropriate for nodes of the machine. The approach scales well, thereby matching a strength of the architecture. Performance results are given as an ideal (or mean) performance on one node. For example ABCD, mean occupancies in A_1, A_2, and A_3 are available by calculation from previous tables. For A_1 of table T4 there is a mean node occupancy of

$$\frac{4}{4}*6 + \frac{3}{4}*4 + \frac{2}{4}*15 + \frac{1}{4}*2 = 17$$

In practice occupancy times are collected separately at each node and then averaged. The result is the same, but unlike using table T3 or T4, no global information is used until the final averaging. Instrumentation collection demands are consequently much lighter. Attributes A_j take only the values 0 and 1. Let $o_i(1)$ denote the occupancy time of A_i at value 1. Total elapsed time T is $T = o_1(1) + o_2(1) + o_3(1)$, since a node of system ABCD is always in one (and only one) of the A_i. Consequently, $T - o_i(1) = o_i(0)$, and only the $o_i(1)$ need be recorded. Averaging across nodes for means:

$\bar{o}_1(1)$	$\bar{o}_2(1)$	$\bar{o}_3(1)$
17.0	11.5	0.5

T5: Mean Node Occupancies for A_1, A_2, A_3

Measuring or Estimating. Simple reasoning about states leads to the above. While measurements taken at one randomly selected node could also establish occupancy estimates, these entail more uncertainty: Overall service is *estimated*, based upon the chosen node being fully representative. Measuring the *full application service* by summing local measurements ensures that incidental variations among nodes have much diminished influences. Consequently, one role of any line of argument is to show what system state details have been lost or preserved, and at what cost. As for mean node occupancies (*e.g.*, in T5), homogeneous applications occur frequently enough to render the approach worthwhile *provided that it is accurate*. Three general examples provide evidence that it is.

Mean Node Occupancies and Three Hypercube Paradigms

The experiments with local states involve our NIST specially-instrumented iPSC-1 Hypercube system. A choice of three synthetic communication benchmarks underscores that (i) iPSC-1 node performance is relatively simple, and (ii) inter-node communication is a dominant performance factor. The benchmarks explore points along a spectrum of communication interdependencies [LYO89b]. In *Random*, tasks proceed independently once each is spawned. Radiation transport has this property. *Mesh* simulates locally-dependent processes typical of fluid models. *Ring* has each node dependent upon all others for the next time-step calculation. Molecular dynamics might approach this global level of process dependency. The analysis yields a fresh look at original data gathered in 1989 by R. Snelick [SNE89] to the following specifications:

	Application Paradigms		
	B1--Random	**B2--Mesh**	**B3--Ring**
Distinct Trial Parameter Sets	60	75	75
Range, Elapsed Times	8.4--202 s	4.2--146 s	3.1--789 s
Independent Variables	Ranges of Independent Variables		
exit rate, e	75--225	n.a.	n.a.
nodes, n	4--16	9--16	4--16
packets/msg, p	fixed (1)	1--16	1--16
grain, g	200--1000	20--100	20--100

Specifications for Benchmark Trials [from SNE89]

There are $m=4$ dependent variables (observables) that identify critical states of each node: communication interrupt service (I), remainder of system service and user-mode (U), message-sending (S), and message-receiving (R). These are the A_i. (An iPSC-1 node has no separate communication processor.) While further details of node state would be informative (such as *synchronous* or *asynchronous* communication), the four rudimentary states are quite serviceable. Our 16 processor system has 16 corresponding instrumentation boards that capture node state information.

Mean node state occupancies for I, S, R, and U are recorded along with the independent parameter settings. For example, *Ring's* independent parameters are n, p, and g. These respectively denote the number of nodes, packets per message, and computational grain, *i.e.*, DO-loop iterations per message datum. Trials for *Ring* are each recorded as:

Independent Variables			**Dependent Variables (Observables)**			
n	p	g	$\bar{o}_1(1) \equiv I$	$\bar{o}_2(1) \equiv S$	$\bar{o}_3(1) \equiv R$	$\bar{o}_4(1) \equiv U$

Records for *Mesh* are the same as for *Ring*. *Random* omits p, which is fixed at one packet per message, and introduces e, the *exit rate*. The exit rate controls how often a message is sent to another randomly-chosen node. Each such message contains some workload specification (again a random amount). While the benchmarks B1-B3 can be compared solely on the basis of state occupancies established from one run per benchmark, the common independent variables (e, n, p, g) support a far broader range of evaluations. Consequently, each benchmark code is run with numerous parameter settings.

Discussion and Results

The 210 (60 + 75 + 75) trial data records have been reduced to a compact tableau (shown below) of four equations per paradigm. An excellent compression of 840 raw measurements (210 trials * 4 states) has been made into 12 short, parametric equations (3 paradigms * 4 states). The reduction is 70:1 for measurement records to equations, a clear demonstration of the efficacy of the state-based approach. There is a moderate but deliberate loss of accuracy. Since tolerances of ±20% are not unusual in everyday software experience, statistical analyses of residual terms were terminated near this tolerance. The number of equations per benchmark corresponds to the number of system states. Tighter tolerances add further terms to the equations, but neither this nor additional data affect the number of equations.

Mean node states I, S, R, U, and their occupancy estimators lend structure to the table. *Columns* sum to elapsed application time. *Rows* provide comparisons among benchmarks for a selected state. For example, scaling the system up in nodes, n, shortens receive times, R, for B1 and B2 but lengthens B3's. Because the applications share several independent parameters, their equations each generate *response surfaces* that can be compared in varying degrees. The surfaces help determine the suitability of benchmarking regions. Certainly parameter settings are suspect when they lie near singularities in the surfaces.

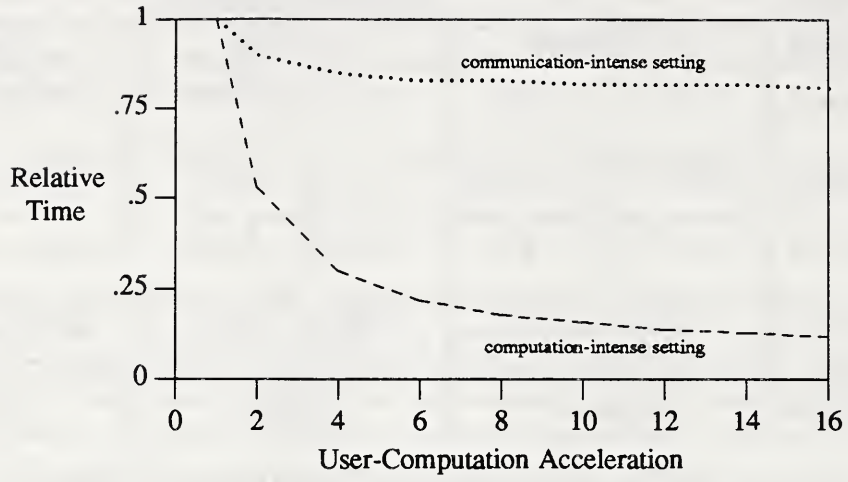
In general, parameter sets of two distinct benchmarks may vary widely. Benchmarks whose parameter sets relate as strictly monotone are easily compared (as here), since a unique mapping from one set of parameter values to the other is ensured. The chosen family, B1-B3, was designed from the start to explore system communication parameters, so parameter commonality is not accidental. A weaker relationship between parameter sets indicates algorithms with less common structure. Some parametric dimensions may lack mutual meaning. Even among the tailored set, parameter e is unique to B1. An *ad hoc* assemblage of benchmarks may have no significant parameter relationships, a fact that weakens its utility. What benchmarks always share are the performance states for the system of interest. Occupancy signatures, at least, lie on the same system dimensions. In worst case, a summary is a table of static numerical entries, rather than a tableau of dynamic parametric equations. Yet, even the static table has considerably more consistency and detail than the usual benchmark results.

iPSC States	Application Paradigms		
	B1--Random	**B2--Mesh**	**B3--Ring**
Comm. Intrpt., I	$0.00227\,\frac{e}{n} + 0.040$	$0.952\,\frac{p}{n} + 0.010$	$0.00760\,pn + 0.0660$
Send, S	$0.0013\,\frac{n+e}{n} - 0.0188$	$2.40\,\frac{p}{n} + 0.333$	$0.0114\,p + 0.067$
Receive, R	$22.6\,\frac{g-100}{ne} + 2.86$	$0.619\,\frac{p}{n} + 0.300$	$0.0367\,(p+4)n + .296$
User, U	$0.522\,\frac{g}{n} + 0.905$	$0.089\,pg + 40.0\,\frac{p}{g}$	$0.0299\,png - 0.380$
Estimator T=I+S+R+U Accuracy, 95% Confidence	*± 15% of measured elapsed time*	*± 10% of measured elapsed time*	*± 20% of measured elapsed time*

Comparing Benchmarks B1-B3 via their State Occupancy Estimators
(sum each column for elapsed time)

Problems in the Data. For the three examples, the choice of *grain g=0* is unrepresentative of other grain values. *Zero-grained trials have been excluded.* Another problem of lesser magnitude is the variation of performance against mesh node points, $n = 9, 12, 16$, for *Mesh*. A plot of residuals for this benchmark shows that *Receive* and *Interrupts* have distinct modalities for different numbers of processors. Nonetheless, such variations have been omitted as equation terms above because their effects lack significance. However, identifying sources of uncertainty encourages a more enlightened use of a benchmark. In particular, sensitive parameters are identified and noted. On new architectures these parameters should be checked, for the magnitude of their contributions may differ.

Estimators and Modeling. Besides the powerful compression of information, there is real advantage in having estimator equations that capture benchmark performances via parameters: Their structure supports broader uses than are typical of conventionally-reported results. Take for example *Ring*, whose user-time is a function of grain, g. Grain corresponds to application computation per message packet, so that conjectures on the effects of a user-state accelerator or improved application code can be couched in terms of g. The question is natural to a user. Results appear below. The outcome from varying g is consonant with that obtained from a NIST emulation method; the latter permits variable physical transport speeds on hypercube communication links [AS90]. In both methods, the balance is changed between compute-speed and communication-rate. Also with both, those settings of *Ring* that provide speedup for one case do little in the other. Although convenient and quick, the benchmark method will not always work, for the independent variables may be arranged poorly in the equations for certain questions.



**Relative Improvements for Ring Program,
Two Distinct Parameter Settings**

On a Real System. Everyday use of a computer should be rather removed from the expediencies of experimental methods on test systems. Instrumentation for a commercial system is certainly more convenient when it is automatic within the operating system. Application code is then untouched, and users untroubled by collection details. An operational hypercube system can do this, *i.e.*, it can provide the utility of local collections (as discussed above) at very little expense to user or system. Process context-switching presents a major overhead to which measurement instrumentation overhead is only a small increment. The four iPSC node states *I*, *S*, *R*, and *U* need a fast system clock and fast registers to record occupancies. Saved context must support these registers. This is roughly twice as much timing context (from twice as many states) as that which supports time-sharing statistics, but it is still slight relative to the overall process context that is saved. The operating system collects and aggregates node statistics upon termination of a job.

Local State Information from Amorphous Systems

The treatment of state is potentially more difficult whenever there are many system factors, each with distinct rates. A static layout of a heterogeneous program will then determine a unique set of resource requests, each of which counts in determining performance variation. Comparabilities among programs will be difficult, because each will have costs bound strongly to its program structure. The best success is when disparate applications can have their resource demands reduced to a limited set of common system resources. However, some recent distributed-memory systems have dynamic allocations. As an example, the Myrias SPS-2 [PPG90] presents a global view of its address space, with page faults generating fetches from other processors. Memory space management is transparent to users. Tasks similarly migrate from one processor to another to improve load balancing. The result is a shifting set of execution costs that is summarized for a job at any point in time via an *estats* invocation:

estats(1):

User	System	Wait	Idle

The items reported are similar to those just seen for the iPSC-1, but factored more conventionally. *System* on the Myrias covers aspects of service in the system-state. *Wait* is time spent awaiting event completions, *e.g.*, page fetches. *Idle* records times when both *ready* and *blocked* queues are empty; because a Myrias task is not bound to a processor, it is possible for a processor to have absolutely nothing scheduled. (The load-balancing mechanism constantly tries to prevent this.) The Myrias *estats* call summarizes many variable memory and task costs that would be too complex in detail. Constant "stirring" by system management removes many concerns about homogeneity or heterogeneity of programs. As with the homogeneous codes on the iPSC, *estats* and its statistics scale up with the system.

Summary and Conclusion

A state occupancy view of performance removes time as an overwhelming concern. While transient behavior not incorporated explicitly into a state is lost, occupancies still capture general performance details. Emphasis is upon comprehensive results that are nonetheless simply compared. Summaries are defined by the resolution of the chosen state-space, rather than the amount of collected data.

The challenge of state occupancies is to move effectively from first principles to actual circumstances. Practical limitations almost always dictate simplifications and approximate methods. This explicit reconciling of theory to practice also identifies sources of measurement uncertainty and error that are so often overlooked in benchmarking. Specific issues involve:

- assuring that states are succinctly defined and recorded,
- managing a large number of states as parallel systems scale up in number of processors,
- compressing state representations,
- obtaining global information or circumventing the global state view.

The issues are related one to another. Certainly, the discussions on homogeneous *applications* and amorphous *systems* hint at typical tradeoff possibilities. From the system standpoint, trends are toward: (i) duplicating via VLSI--*good* because architecture is more uniform; (ii) decentralizing functions--*poor* for ascertaining state; (iii) transparent balancing and referencing--*good* to counter application bindings. This assessment assumes an instrumentation geared primarily toward *time–based samplings*. There will be ample opportunity to break from this mold as systems increasingly employ *by–value* communications, rather than the *by–reference* of shared-memory. As emphasized above, the very philosophy of state occupancies is to banish as much of time (*e.g.*, state sequences, algorithmic iterations) as possible. The important correlations identify which resources handle which demand; exact occurrences in absolute time are immaterial. Consequently, the use of *tagged–messages* would seem to be a natural matching of the strengths of distributed systems and the relaxed requirements of state occupancies. The challenge would be to keep the tagging independent of specific applications. Otherwise, comparisons among applications would be very limited.

Conclusion. Restricting comparisons among applications to either (i) their signatures of system service, or (ii) parametric sets of signatures, is far simpler and more readily accessible than detail-by-detail algorithmic comparisons. Relying principally upon measurement, state occupancies present an attractive complement to other more analytic and specialized methods.

Citations

[AS90] Antonishek, J.K. and Snelick, R.D. "Emulation through Time Dilation." The Fifth Distr. Memory Comput. Conf. (DMCC5) Proc., Charleston, SC 1990, 8pp. (1990).

[FEL64] Feller, W. **An Introduction to Probability Theory and its Applications, Vol. I.** John Wiley & Sons, New York, 1964, p. 36

[FJL88] Fox, G.C., Johnson, M.A., Lyzenga, G.A., Otto, S.W., Salmon, J.K., and Walker, D.W. **Solving Problems on Concurrent Processors, Vol. I.** Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[LS90] Lyon, G.E., and Snelick, R.D. "Workloads, Observables, Benchmarks and Instrumentation." Joint Int. Conf. on Vector and Parallel Processing Proc., (Springer-Verlag, Sept. 1990, Zurich), 86-97.

[LYO89] Lyon, G.E. "Capacity-and-Use Trees for Estimating Computer Performance Variations." Int. Conf. on Computing and Information Proc., Vol. II, (Canadian Scholar's Press, May 1989, Toronto), 309-313.

[LYO89b] Lyon, G.E. "Design Factors for Parallel Processing Benchmarks." Theoretical Computer Science 64, (1989), 175-189.

[LYO90] Workshop I: Performance Evaluation and Benchmarks, G.E. Lyon, Organizer. ACM Computer Architecture Workshops, May 26-27, 1990, Seattle, Washington.

[MCNR90] Mink, A., Carpenter, R., Nacht, G., and Roberts, J. "Multiprocessor Performance-Measurement Instrumentation." IEEE Computer 23, 9(September 1990), 63-75.

[PPG90] Parallel Programmer's Guide. Myrias Computer Corporation, February 1990.

[SNE89] Snelick, R.D. *NIST—Internal Data File, Summer*, 1989 These files summarize several thousand data points for three communication benchmarks running on the iPSC.

[SSM89] Saavedra-Barrera, R.H., Smith, A.J., and Miya, E. "Machine Characterization Based on an Abstract High-Level Language Machine." IEEE Trans. on Computers 38, 12(Dec. 1989), 1659-1679.

NIST-114A
(REV. 3-89)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4418

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
OCTOBER 1990

4. TITLE AND SUBTITLE
State Occupancy Information
for
Performance Comparisons

5. AUTHOR(S)
Gordon Lyon

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)
U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

10. SUPPLEMENTARY NOTES

[ ] DOCUMENT DESCRIBES A COMPUTER PROGRAM; SF-185, FIPS SOFTWARE SUMMARY, IS ATTACHED.

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

A state-based performance characterization attaches fixed processing rates to each service state. However, the number of states can be large. Over time, the sequences of such states are enormous. Consequently, counts of active (but interchangeable) system elements define macrostates, which are fewer. Furthermore, only the occupancy levels of macrostates are recorded. This removes time sequencings as a combinatoric problem, but still captures general performance details. Applications can be compared via the method.

The hypercube and other distributed-memory systems bring both opportunities and challenges to a state-based approach. Certainly processor and communication activities are more easily identified and more independent with distributed-memory than with shared-memory. But isolated nodes also entail problems in capturing global observation states. However, many hypercube application codes are homogeneous across nodes, a fact that makes partial states quite useful. Three paradigms illustrate homogeneous applications with communication dependencies that are strong (global), moderate (local), or weak (uninhibiting and random). The three performance summaries are compact and accurate.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

applications; comparisons; distributed-memory; occupancies; performance; states

13. AVAILABILITY

[X] UNLIMITED

[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).

[ ] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.

[X] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
16

15. PRICE
A02

ELECTRONIC FORM