

Parallel Performance Visualization Using Moments of Utilization Data

T. J. Godin, Michael J. Quinn and C. M. Pancake
Department of Computer Science
Oregon State University
Corvallis, OR 97331 USA

Abstract

We propose a new parallel performance visualization scheme, based on a simple moment analysis of processor utilization data. This method combines the scalability advantages of statistical summaries with the more revealing processor utilization information of Gantt charts. It scales well to large numbers of processors, requiring only storage constant in execution time.

1. Introduction

Processor idling can strongly affect the performance of parallel code. Consequently, many parallel performance tools [2, 4, 7, 9, 11] include one or more utilities, such as Gantt charts [3] or statistical summaries, to measure and display processor utilization information.

Statistical summaries provide integrated totals of processor use [5]. Summary-based tools scale well to massively parallel computations, since they store and display only a few parameters per processor. However, summaries reveal little about the time dependence of resource use. A programmer, for example, might be able to identify under-used processors, but not the times when those processors were idle.

On the other hand, a utilization Gantt chart (UGC) provides a detailed temporal map of processor activity, showing times and durations of CPU idle states. UGC analysis, however, is less scalable to large computations [10]. A UGC incorporates large amounts of profile data that must be stored in primary or secondary memory. The resulting drain on system resources can substantially perturb the computation being measured. Moreover, a large UGC display is visually dense. It may contain much information that is not useful. For example, a user may wish to concentrate on a few large patterns of inactivity, rather than many small, uncorrelated gaps.

In this paper we propose a parallel performance visualization scheme that combines the main advantages of summaries and UGCs. Our approach is to partially transform each processor's utilization data using a simple moment analysis. The transformation produces a small set of parameters that characterizes the most important features of the UGC.

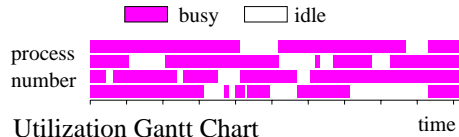


Figure 1. Processor activity as a function of time.

This method can convey a great deal of useful performance information about a large number of processors, in a visually economical way. Moreover, the collection and processing of temporal moments is computationally efficient, requiring relatively little storage, computation, or I/O. Moment analysis therefore offers the scaling advantages of statistical summaries, while retaining substantial temporal resolution.

2. What Is a Gantt Chart?

Before proceeding, it is helpful to review just what a Gantt chart is. We can think of a UGC as a temporal *distribution* of processor use. That is, for each of i processors, we define a function $g_i(t)$ such that

$$g_i(t) = \begin{cases} 1 & \text{if processor } i \text{ is busy at time } t \\ 0 & \text{if processor } i \text{ is idle at time } t \end{cases} \quad (1)$$

where t represents some temporal quantity (wall clock, CPU cycles, etc.). A graphically "synchronized" set of $g_i(t)$, one per processor, characterizes overall utilization (Fig. 1).¹

Now, the key idea is that each g_i is a function of *time*. A statistical summary is not. This is what can make a UGC so much more useful than a summary. A summary provides only a time integral of each $g_i(t)$ (*how much* idle/busy time accrues during execution). By contrast, the shapes of the g_i , with respect to the variable t , reveal not only the total idle time, but *when* idle states occur [5, 10].

This wealth of additional information, however, implies a significant drawback. To construct a detailed UGC a tool must record, during run time, each processor's state

¹ To keep this discussion relatively simple, we will consider only two-state (idle/busy) distributions here. That is, we will not consider task Gantt charts or distributions with an "overhead" state.

for a large sample of distinct t values. The size of the resulting data set scales linearly with execution time.

Our goal is to construct a performance display which preserves important time-dependent information from the $g_i(t)$, without storing or displaying a large number of parameters. To accomplish this, we propose a rethinking of how time dependent utilization data are presented.

3. Thinking in Moment Space

The important features of a distribution can often be characterized by a relatively small set of parameters. For example, if you are given the following representation of a Gaussian distribution:

$$\begin{aligned} \text{norm} &= N \\ \text{mean} &= \bar{x} \\ \text{variance} &= \sigma^2 \end{aligned} \quad (2)$$

you can visualize what that distribution looks like. You do not need to use its explicit representation,

$$f(x) = \frac{N}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}}, \quad (3)$$

to compute and plot each value of f for a large sample space of x values. Our experience in statistical analysis has trained us to infer where f is peaked, and how wide and high the peak is, from knowing only the values of N , \bar{x} , and σ^2 . These three parameters are called *statistical moments* of f [1].

Equation (3) is a representation of f in x space; it explicitly specifies a value of f for every value of x on the interval $[-\infty, \infty]$. We visualize Gaussians, however, by thinking in the *moment space* of Eqn. (2), in which the representation of f has a domain of only three parameters. The two representations are related by a transformation.

Since f is Gaussian, the transformation is exact. That is, the values of N , \bar{x} , and σ^2 completely specify the shape of f in x space. To *exactly* specify a more complicated distribution (e.g., a Gantt chart), many more parameters may be required. A Gantt chart, however, can be *approximately* specified using a small number of parameters.

4. Moment Representation for Gantt Charts

We now introduce a simple moment-space representation for UGCs. That is, we define a set of four parameters, simply related to statistical moments, that approximately describe the time dependence of processor activity.

At this point it is essential to concentrate on what each parameter says about the shape of an underlying UGC, rather than on mathematical detail. Consequently, the following description is heuristic and visual. Section 7 will present a more formal mathematical justification.

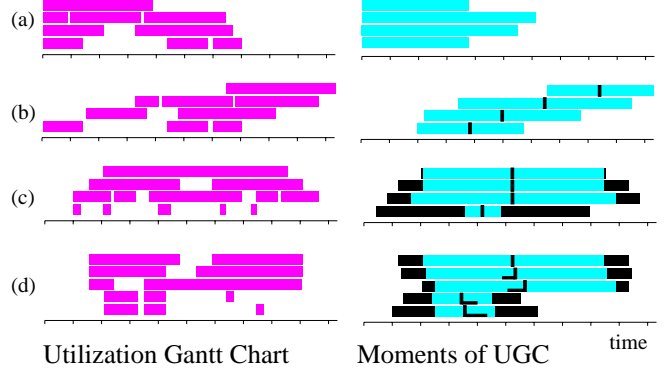


Figure 2. Graphical representation of scaled moments. (a) The norm is represented by a grey bar of length m_0 . (b) The mean is represented by centering the m_0 bar at time m_1 , and placing a black tick mark there. (c) The scaled deviation is the distance from the m_1 tick to the ends of the outer black bars. (d) Skew is represented by a thin black line of length m_3 , extending from m_1 .

Consider a computation which begins at time t_0 and ends at time t_f . Suppose that during this computation, one processor's Gantt chart is represented by a function $g(t)$, defined as in Equation (1).

We define the *zeroth moment* of the distribution $g(t)$ as

$$m_0 = \int_{t_0}^{t_f} g(t) dt. \quad (4)$$

This is the *norm* of $g(t)$. It equals the total amount of time the processor is active. Statistical summaries typically provide m_0 . Note that this parameter has units of time, and is always nonnegative. We can therefore graphically represent its value by a bar of length m_0 , plotted along a time axis (Fig. 2a).

The *first moment*,

$$m_1 = \frac{1}{m_0} \int_{t_0}^{t_f} t g(t) dt, \quad (5)$$

is simply the statistical mean of the distribution $g(t)$. It is large when most of a processor's activity occurs close to t_f ; it is small when most activity occurs close to t_0 . It therefore measures the time at which the processor was most likely to be active. Moment m_1 , like m_0 , has units of time. We can represent m_1 graphically by positioning the center of the m_0 bar at time $t = m_1$, and placing a black tick mark at that point (Fig. 2 b).

We call the *scaled second moment* of $g(t)$,

$$m_2 = \sqrt{3\mu_2}, \text{ where } \mu_2 = \frac{1}{m_0} \int_{t_0}^{t_f} (t - m_1)^2 g(t) dt, \quad (6)$$

the *scaled deviation* of $g(t)$. It, too, has units of time. It is simply the standard deviation of the distribution, multiplied by a factor of $\sqrt{3}$. (Section 7 discusses the motivation for this scaling factor.) Recall that a deviation indicates the amount of "spread" in a distribution. So, for $g(t)$, it defines a time range,

$$m_1 \pm m_2, \quad (7)$$

which will likely encompass the bulk of a processor's busy time. Note that this range has width $2m_2$, and is centered about the mean. We will show (Section 7) that in all cases,

$$2m_2 \geq m_0. \quad (8)$$

In other words, on our display the width of the range in Eqn. (7) will always be greater than or equal to the width of the m_0 bar. Moreover, both ranges are centered at the mean. This suggests a simple way of plotting the scaled deviation: we simply place black bars on either side of the gray m_0 bar, extending out to the ends of the deviation range (Fig. 2c).

Moment m_2 takes on its smallest possible value ($m_2 = m_0/2$, hairline black bars on the moment graph) if $g(t)$ is composed of one unbroken "busy" state. It is larger ($m_2 > m_0/2$, long black bars on the graph) if the processor has more than one busy state, interrupted by idles.

Finally, the *scaled third moment* of $g(t)$, or *scaled skew*, is given by

$$m_3 = 3\mu_3^{1/3}, \text{ where } \mu_3 = \frac{1}{m_0} \int_{t_0}^{t_f} (t - m_1)^3 g(t) dt. \quad (9)$$

Moment m_3 also has units of time, and can be positive or negative. A small skew indicates that $g(t)$ is roughly symmetric about m_1 . Intuitively, busy time is equally distributed along the t axis. A large skew indicates asymmetry in $g(t)$; the processor had more total activity earlier than time $t = m_1$ than later, or vice versa. Positive skew implies that most work occurred prior to m_1 ; negative skew implies the reverse. We plot skew as a thin line extending from the m_1 tick mark, whose length equals m_3 (Fig. 2d).

Table I summarizes the properties of the four moments; Fig. 3 illustrates our corresponding display scheme. Note what the various moments have in common. Each is a scalar, with units of time, so each can be plotted as a bar or line along a time axis. Each tells us something about the shape of the function $g(t)$, which consequently reveals something about the temporal distribution of processor activity. Note especially how each Gantt chart variation in Fig. 2 "translates" into a corresponding moment variation.

The lowest order moment, m_0 , gives a very approximate picture of $g(t)$, equivalent to that provided by a

	<i>range</i>	<i>indication</i>
m_0	$0 \leq m_0 \leq (t_f - t_0)$	small: processor relatively inactive large: processor relatively active
m_1	$0 \leq m_1 \leq (t_f - t_0)$	small: most processor activity occurs early in computation large: most processor activity occurs late in computation
m_2	$m_2 \geq \frac{m_0}{2}$	$m_2 = \frac{m_0}{2}$: processor's Gantt chart is a single, uninterrupted "on" state $m_2 \gg \frac{m_0}{2}$: processor has frequent or long interruptions
m_3	$m_3 < (m_1 - t_0)$ $m_3 < (t_f - m_1)$	$m_3 = 0$: $g(t)$ is symmetric about m_1 $m_3 > 0$: majority of busy time occurred prior to m_1 $m_3 < 0$: majority of busy time occurred later than m_1

Table I. Summary of scaled moments.

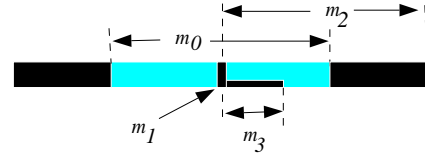


Figure 3. Display scheme for scaled moments.

statistical summary. Adding each new moment gives a successively more accurate picture of $g(t)$. If a user is provided with a set of moments found using Eqns. (4), (5), (6), and (9), and understands what each moment implies about the structure of $g(t)$, he or she can roughly infer the structure of $g(t)$ via a mental inversion of the moment transformation.

In theory, we need not stop at four moments. We could continue to define fourth, fifth, and higher order moments, progressively refining the picture. We have stopped here, however, for two reasons. First, we wish to keep this discussion, and our display, relatively simple. More importantly, the four moments defined here are sufficient, in many cases, to reveal a great deal of useful performance information. The following section demonstrates this through examples.

5. Moment Analysis In Practice

The central idea of a performance evaluation tool is to inform a user when something is going wrong, and provide some idea of where the problem is. To show how a moment analysis display can accomplish this, we turn to some examples.

First, consider the UGC in Fig. 4. This is an approximate rendering of profile data presented in a demonstration of the Convex CXtrace™ performance tool [2]. (CXtrace™ provides a task Gantt chart with additional overlaid data; we simplified the demonstration display to

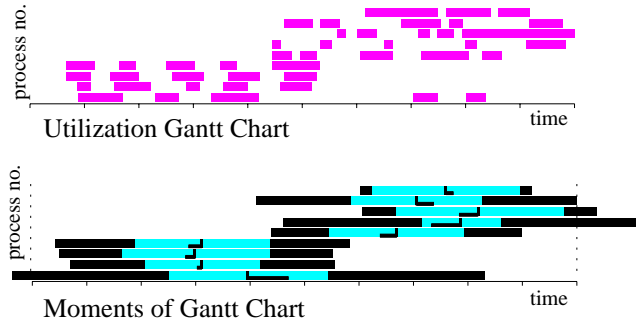


Figure 4. (a) Example of trace data adapted from [2]. (b) Same data redisplayed using the moment analysis described in the text.

moment	size	indication
m_0	small	low total activity
m_1	small	activity concentrated mainly in early part of computation
m_2	large	significant idle time within active period
m_3	large, positive	small segment of activity late in computation

Table II. Moments of utilization data for the lowermost processor in Fig. 4(b).

create a UGC, approximately corresponding to the same profile data.) Alongside, we display the first four moments for each processor.

This simple example bears close study. The moments clearly reveal that about half of the processors are active primarily in the early phase of computation; the other half are active primarily in the later phase. The top two bars represent, respectively, a processor busy for a short period, interrupted only by short idle breaks (note small variance), and a processor with more idle time (larger variance), concentrated somewhat toward the end of its overall active period (larger, positive skew).

Note how each moment, for each processor, can be related to an important feature in the UGC. As an example, Table II provides a detailed comparison of Gantt chart and moments for the lowermost processor in Fig. 4. A glance at the UGC bears out each observation. In fact, by comparing each processor's moments with the examples in Fig. 2, one can easily infer most of the important features of this UGC.

It is important to note that despite the visual similarities between the two panels in Fig. 4, the moment display actually presents far fewer parameters than the UGC. This is more apparent in our second example, Fig. 5, which shows a more complicated UGC. It is an approximate representation of a chart originally presented as a demonstra-

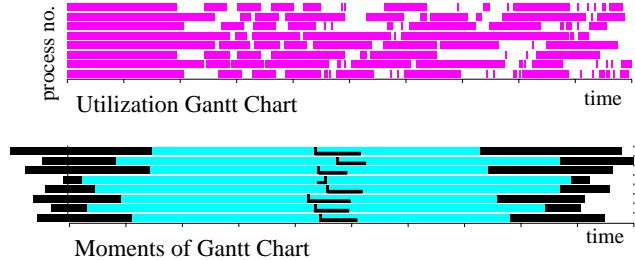


Figure 5. (a) Example of trace data adapted from [4]. (b) Same data redisplayed using moment analysis.

tion of the ParaGraph performance tool [4]. (ParaGraph distinguishes between "busy" and "overhead" states; for simplicity we classify both states as "busy" here. In the example, the total amount of overhead is small.)

Like Fig. 4, it is easy to infer from the moments which processors are less active. One can also infer some information about the distribution of idle time. Note, for example, the lowest two processors, which have similar mean and skew but different norm and deviation. This indicates that both processors do more work early in the computation, but the lowermost has much larger idle gaps late in the computation.

More importantly, the UGC must be displayed on a wide time scale, to accommodate its dense structure within a typical computer screen resolution. For consistency, we have plotted the moments on the same time scale. Clearly, however, the moments could be plotted on a much narrower time scale without loss of information.

6. What's the Catch?

Clearly, such a simple moment display cannot reproduce all of the structure of a dense Gantt chart. Much detail about the location of processor idle times, and their synchronization with idle states of other processors, will be lost. In some cases this may be significant.

In a system of several thousand processors, however, we question the extent to which most of this information will actually be used in performance tuning. In fact, for more than 100 processors a UGC usually contains too much information to be useful [10].

By contrast, a moment display shows only a few parameters per processor. Yet, even this very simple representation conveys significant information about the length and time of idle states, which will likely have overriding importance in a first optimization "pass." Successive tuning runs can use more moments, or perform a similar moment analysis on a smaller time interval, to obtain more specific resolution of idle states. If final optimization requires still finer time resolution, a user could eventually resort to a full UGC analysis.

Finally, and most importantly, a moment analysis has scaling properties far superior to those of a full UGC

analysis (see Section 8). It is therefore less likely to perturb the primary computation. In this sense, the moment display may depict true system performance more accurately than a full UGC.

7. Mathematical Basis of Moment Analysis

Mathematically, computing statistical moments of a distribution is a transformation. Transformation analysis provides a formal, quantitative means of characterizing the shape of a function or distribution in a concise, consistent way. The detailed theory behind such an analysis (completeness, estimation of convergence, etc.) is beyond the scope of this paper. Here, we will highlight some important points that relate to our performance display scheme.

Suppose we wish to characterize the shape of some function $f(x)$. The idea behind a transformation is to successively compare $f(x)$ to a sequence U of basis functions, where $U = \{u_0(x), u_1(x), u_2(x), \dots\}$. We first find out how much $f(x)$ "looks like" $u_0(x)$, then find out how much $f(x)$ "looks like" $u_1(x)$, etc. The measure of "alike-ness" between $f(x)$ and a basis function $u_i(x)$ is called the *overlap* of $f(x)$ with $u_i(x)$, and is given by the formula

$$\mu_i = \frac{1}{N} \int f(x) u_i(x) dx. \quad (10)$$

The parameter μ_i is also called the *projection* of $f(x)$ onto $u_i(x)$. N is some normalization constant.

The choice of basis set U usually depends on a particular application. For example, a Fourier transform uses a basis of sine and cosine functions; the resulting μ_i are called Fourier coefficients. In Sections 3 and 4 we use a different basis set, given by

$$\begin{aligned} u_0(x) &= 1 \\ u_1(x) &= x \\ u_i(x) &= (x - \mu_0)^i, \quad i = 2, 3, 4, \dots \end{aligned} \quad (11)$$

Each μ_i generated by (11) is called the i th statistical moment of f . The lowest order moments are called, respectively, the norm, mean, variance, etc. of a function.²

For two reasons, we have chosen Eqn. (11) to serve as a basis of our analysis. First, most users have some statistics training, and are therefore familiar with the geometric interpretations of the resulting moments. They know, for example, what a large variance or skew implies about the shape of a distribution. In the language of Section 3, users already know how to "think" in the moment space defined by Eqn. (11). Second, computing this set of moments has favorable complexity (see Section 8).

² If f is Gaussian, all moments produced by (11), except the first three, always vanish. For other types of distributions, however, higher order moments will, in general, be nonzero [1].

Applying the integral in (10) to a UGC is straightforward, since the distribution of Eqn. (1) can only take on values zero or one. If a processor has n busy states, and the j th busy state begins at time a_j and ends at time b_j , then Eqn. (10) reduces to

$$\mu_i = \frac{1}{\mu_0} \sum_{j=1}^n \int_{a_j}^{b_j} (t - \mu_1)^i dt. \quad (12)$$

To provide more intuitive meanings, we have scaled the moments as follows:

$$\begin{aligned} m_0 &= \mu_0 & m_2 &= \sqrt{3}\mu_2 \\ m_1 &= \mu_1 & m_3 &= 3\mu_3^{1/3} \end{aligned} \quad (13)$$

Note that m_2 is the standard deviation, multiplied by $\sqrt{3}$. The constant is included for the following reason. Suppose $g(t)$ consists of a single "busy" bar of length m_0 . That is, a processor has one, and only one, busy state lasting from time t_1 to time t_2 . Then,

$$m_0 = t_2 - t_1 \quad \text{and} \quad m_1 = \frac{t_2 + t_1}{2}. \quad (14)$$

Inserting these into the integral

$$\mu_2 = \frac{1}{m_0} \int_{t_1}^{t_2} (t - m_1)^2 dt \quad (15)$$

yields

$$\mu_2 = 3m_0^2. \quad (16)$$

Finally, inserting (16) into (13) yields the result $m_0 = 2m_2$, which we cited in Section 3. With this factor, the deviation is always larger than the norm, allowing us to plot deviation as a line extending past the m_0 bar.

The scaling of skew is arbitrary. We have selected a constant which scales a typical skew value to a visually useful length.

8. Computational Complexity

The computational complexity of a performance evaluation tool is a crucial issue. One would like the performance of the primary calculation in the presence of the tool to closely approximate its performance in the tool's absence [6, 8, 10]. Consequently, a tool is more useful when it consumes fewer system resources (memory, CPU cycles, I/O, etc.).

Moment analysis is relatively unobtrusive, in the sense that it consumes relatively few resources during the primary computation. The following two observations make this apparent.

First, note that in each of our examples, we have simply applied a post-mortem analysis to an existing UGC. So, in the worst case, a moment analysis will not perturb system performance any more than computation of a UGC.

Second, we can do much better by computing moments directly, without recording a full UGC. Recall (Section 7) that computing a moment requires evaluating an integral like Eqn. (12). By the binomial theorem, the integrand $(t - m_1)^i$ is an i th-order polynomial in t , leading to an expression consisting of i antiderivative terms of the form $a t^{n+1}/(n + 1)$, where a is a constant. To calculate (12) "on the fly" we need only store and update one value for each term. In other words, the number of stored values equals the number of moments. Hence storage is constant in execution time, an extraordinary improvement over UGC analysis.

9. Conclusion

Each of our examples displays only four parameters per processor. With appropriate coloring, each processor can be represented by a line one pixel thick. Consequently, a 1280-by-1024 screen could easily accommodate a thousand-processor moment display.

What we envisage is a large-scale display, in which the main features of utilization can be easily picked out. A conception of such a display appears in Fig. 6. In the display, note how proximity between data for contiguous processors causes the various moment indicators to merge into "areas" of m_0 , m_2 , etc. A user able to mentally transform back to a t space representation, will be able to identify global patterns of inactivity, that closely correlate to those present if the tool were not used. In the tuning process, a user would progressively modify code to generate a display in which moments such as m_0 and m_1 are more aligned, and moments such as m_2 and m_3 consume less display area.

Appropriate layering or color schemes could be used to include more moments, or information about other types of data (e.g., "overhead" states).

Acknowledgment

This work was supported by Sandia National Laboratories, and by the National Science Foundation under grant ASC-9523629.

References

1. Bancroft, T. A. and Anderson, R. L., *Statistical Theory in Inference and Research*. Dekker, New York (1981).
2. *CXTrace User's Guide*. Convex, Richardson (1994).
3. Gantt, H. L., "Organizing for Work," *Industrial Management* (Aug. 1919), pp. 89-93.
4. Heath, M. T., and Ethridge, J. A., "Visualizing the Performance of Parallel Programs," *IEEE Software* **8**, 4 (Sep. 1991), pp. 29-39.

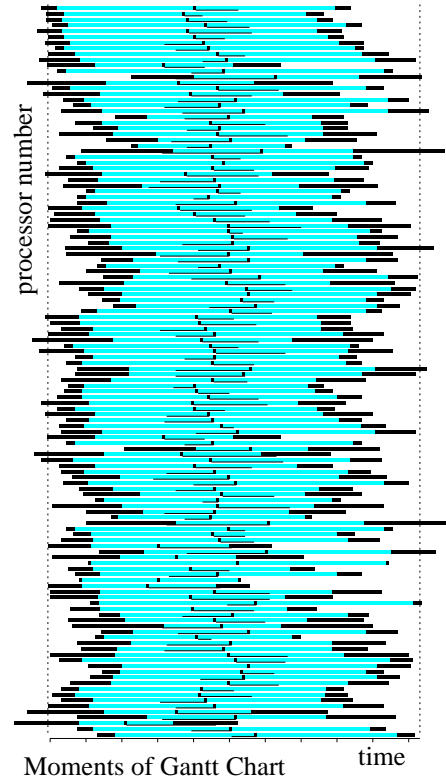


Figure 6. Conception of a moment display for a large number of processors (here, 128). Note how the various contiguous bars combine visually to form lines and areas.

5. Heath, M. T., Maloney, A. D., and Rover, D. T., "Visual Display of Parallel Performance Data," *Computer* **28**, 11 (Nov. 1995), pp. 21-28.
6. Kraemer, E. and Stasko, J. T., "Visualization of Parallel Systems: An Overview," *Journal of Parallel and Distributed Computing* **18**, 2 (June 1993), pp. 105-117.
7. Maloney, A. D., Hammerslag, D. H., and Jablonowski, D. J., "Traceview: A Trace Visualization Tool," *IEEE Software* **8**, 4 (Sept. 1991), pp. 19-28.
8. Pancake, C. M., Simmons, M. L., and Yan, J. C., "Performance Evaluation Tools for Parallel and Distributed Systems," *Computer* **28**, 11 (Nov. 1995), pp. 16-19.
9. Rudolf, L. and Segall, Z., "PIE: a Programming and Instrumentation Environment for Parallel Processing," *IEEE Software* **2**, 6 (1985), pp. 22-37.
10. Tomas, G. and Ueberhuber, C., *Visualization of Scientific Parallel Programs*. Springer-Verlag, Berlin (1994).
11. Yan, J. C., Sarukkai, S., and Mehra, P., "Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs Using the AIMS Toolkit," *Software-Practice and Experience* **25**, 5 (Apr. 1995), pp. 429-461.