

Managing Performance Analysis with Dynamic Statistical Projection Pursuit¹

Jeffrey S. Vetter
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California 94550 USA
j-vetter@llnl.gov

Daniel A. Reed
Department of Computer Science
University of Illinois
Urbana, Illinois 61801 USA
reed@cs.uiuc.edu

Abstract

Computer systems and applications are growing more complex. Consequently, performance analysis has become more difficult due to the complex, transient interrelationships among runtime components. To diagnose these types of performance issues, developers must use detailed instrumentation to capture a large number of performance metrics. Unfortunately, this instrumentation may actually influence the performance analysis, leading the developer to an ambiguous conclusion. In this paper, we introduce a technique for focussing a performance analysis on *interesting* performance metrics. This technique, called dynamic statistical projection pursuit, identifies *interesting* performance metrics that the monitoring system should capture across some number of processors. By reducing the number of performance metrics, projection pursuit can limit the impact of instrumentation on the performance of the target system and can reduce the volume of performance data.

1 Introduction

As high-performance computer systems and applications continue to increase in complexity, the process of performance analysis grows more difficult. Runtime performance problems result from a web of interactions among components including both physical (e.g., cache hierarchy) and logical resources (e.g., I/O buffers). Although the performance implications of each isolated component is relatively well understood, it is less

clear how this web of time-varying relationships influences overall system performance. To understand these relationships and correct performance problems, developers must use performance instrumentation systems that capture detailed information on a large number of these time-varying performance metrics. Unfortunately, this instrumentation can influence the performance of the target system and can produce tremendous volumes of data [13].

To help combat these consequences of performance instrumentation, the instrumentation system should provide support for selecting metrics and measurement points so as to minimize the effects of its instrumentation. As a result, the challenge for developers of performance analysis systems is to reconcile these goals of generating detailed, useful performance information while not dramatically impacting the characteristics of the performance analysis.

We introduce *dynamic statistical projection pursuit* (or PP) as one possible strategy to reduce perturbation and data volume while retaining interesting characteristics of performance data. This statistical technique, when used for performance analysis, attempts to reduce the number of performance metrics (or dimensionality) that a monitoring system must manage, which, in turn, can dramatically reduce perturbation and data volume. Our tool, which uses projection pursuit, achieves this goal by periodically identifying *interesting* performance metrics. The runtime monitoring system can use this information to focus instrumentation on the

¹ This work was supported in part by the Defense Advanced Research Projects Agency under DARPA contracts DABT63-94-C0049 (SIO Initiative), F30602-96-C-0161, and DABT63-96-C-0027 by the National Science Foundation under grants NSF CDA 94-01124 and ASC 97-20202, and by the Department of Energy under contracts DOE B-341494, W-7405-ENG-48, and I-B-333164.

Appears in Proc. SC99, Portland, Oregon, USA (November 1999).

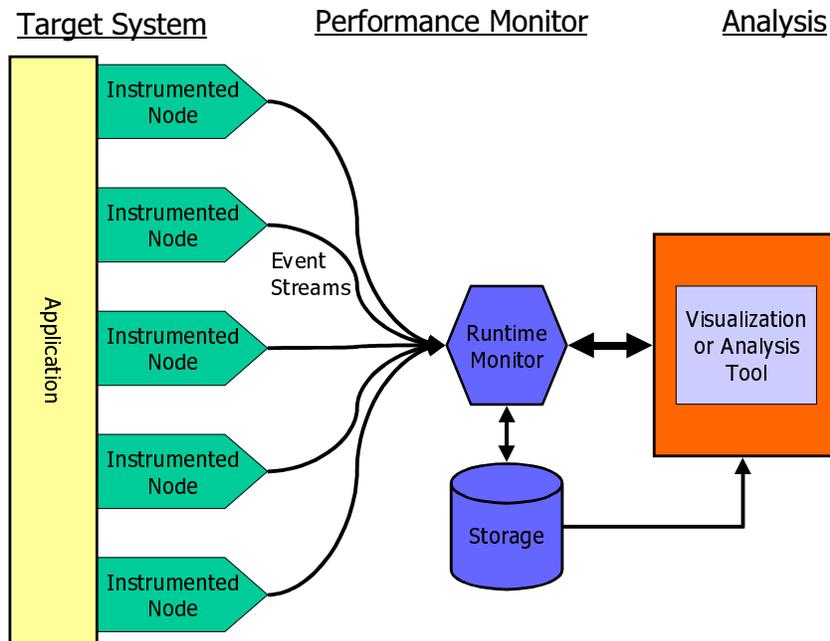


Figure 1: Overview of the generation of performance metrics in a HPC environment.

interesting metrics. By reducing the number of performance metrics, the system can reduce the cumulative amount of data it must manage.

1.1 Framework

Our prototype framework for projection pursuit is event-based performance monitoring. This type of monitoring gathers performance metrics on as shown in Figure 1. Individual processors, whether parallel or distributed nodes, generate streams of events. Each event, in turn, contains a timestamp, event identifier, and some number of metric-value pairs. Subsequently, these events can be either written to files or consumed by a runtime monitoring system. Often, some centralized agent merges multiple event streams in ascending timestamp order for additional analysis or visualization. In this framework, event tracing provides the detailed data needed to understand software component interactions, albeit potentially at great cost. Consider that with P processors generating n metrics as discrete events, the monitoring system must accept data at the rate of approximately $n \times P$ events at any time t . Consequently, over a very short period of time, this data rate can overwhelm most runtime monitoring systems or storage facilities. Thus, an intricate and complex problem emerges: event

tracing is desirable to understand detailed behavior, but the potentially large data volume, large number of performance metrics, and behavioral perturbations make event-tracing impractical for large, long-running applications.

To retain the advantages of event tracing while reducing total data volume and perturbation, one must intelligently manage both the number of metrics needed to identify bottlenecks (i.e., n) and the number of locations where data must be captured (i.e., P).

1.2 Motivation

Two factors motivate our particular investigation of projection pursuit: the necessity to provide additional support to users' analysis of huge performance datasets, and the desire to automate instrumentation management within the runtime monitoring system.

First, we wanted a technique that could provide users with hints about interesting regions of the dataset. Given the size of these datasets, locating anomalies and other patterns can be tedious and error prone for humans. Projection pursuit solves this problem directly by extracting statistically interesting features from the dataset. This information can be mapped to a software visualization or some other analysis tool to aid a

user’s exploration of the performance data.

Second, the ultimate goal of automated performance analysis is to have a runtime monitoring system capable of automatically focussing on important sections of code and performance data. To achieve this goal, the monitoring system must have a suitable, objective policy that it can perform automatically without human intervention. Because projection pursuit is a statistical technique that describes a dataset with objective measures, it is one good candidate for this policy.

1.3 Paper Organization

The rest of this paper discusses these issues in more detail. Section 2 introduces performance metric spaces. Section 3 explains projection pursuit and describes how we use it for performance analysis. Section 4 uses experimental traces to evaluate projection pursuit. Section 0 outlines related work. Finally, Section 6 summarizes our conclusions.

2 Performance Metric Spaces

To formalize this notion and to provide a basis for analysis, consider a set of n dynamic performance metrics, each measured on a set of P parallel tasks. Conceptually, one can then view an event trace as defining a set of n dynamic performance metrics, $m_i(t)$, on each of P tasks

$$(m_1(t), m_2(t), \dots, m_n(t))_p \quad p = 1, 2, \dots, P$$

that describe parallel system characteristics as a function of time t . Following [16], if R_i denotes the range of metric $m_i(t)$, we call the Cartesian product

$$M = R_1 \times R_2 \times \dots \times R_n$$

a performance metric space. Thus, the ordered n -tuples

$$(m_1(t) \in R_1; m_2(t) \in R_2; \dots; m_n(t) \in R_n) \quad (1)$$

are points in $M(t)$, and the event trace defines the temporal evolution of these P points in an n dimensional space.

Figure 2 illustrates a 3-D performance metric space. Each of the ten points represents the value of three metrics for one processor at one point in

time. The collection of these ten points defines a metric trajectory over time. While this trivial example illustrates our formalization, we expect to use our techniques on much larger systems where $n > 10$ and $P > 10$.

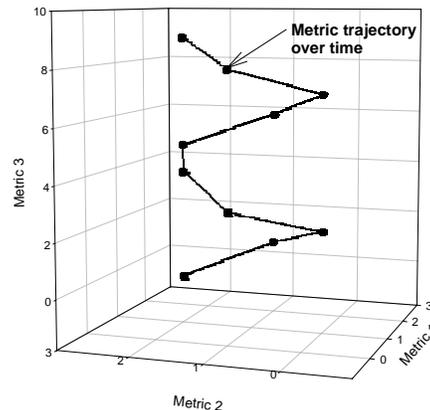


Figure 2: Example of performance metric space. (Single processor with three metrics and 10 sample points.)

The goal of event trace data reduction is now clear—one must reduce both the number of measurement points (i.e., reduce P) and the dimensionality of the metric space (i.e., reduce n). Dynamic statistical clustering [16] has addressed the first problem of reducing the number of measurement points. We propose projection pursuit as a solution to the second problem of reducing the dimensionality of the performance metric space.

3 Projection Pursuit

Projection pursuit [15] is a statistical technique for exploratory data analysis on high-dimension data. It shares many ideas with other techniques such as Grand Tour [6] and Parallel Coordinate Plots [20]. Intuitively, projection pursuit projects many different views of high-dimension data onto a 3-D space and then, it judges the success of this mapping with an objective, mathematical calculation. Upon termination, projection pursuit outputs the particular mapping that provided the most successful view. For instance, Figure 3 illustrates the mapping of 3-D scatterplot data onto two orthogonal 2-D planes. Given these two 2-D projections of the 3-D data, it is easier to judge the data’s underlying structure.

3.1 Background

In this work, we use 3-D projection pursuit; this version maps high-dimensional data into 3-D space and measures the *interestingness* from properties in three dimensions. However, note that projection pursuit is not limited in the number of input dimensions, and recently, researchers have extended PP to higher-dimensions, such as 4-D and 5-D [6]. Furthermore, if the system needs a ranking of all dimensions, then PP can be used repeatedly on one dataset while removing the winning three dimensions after every PP phase and re-executing PP on the smaller dataset.

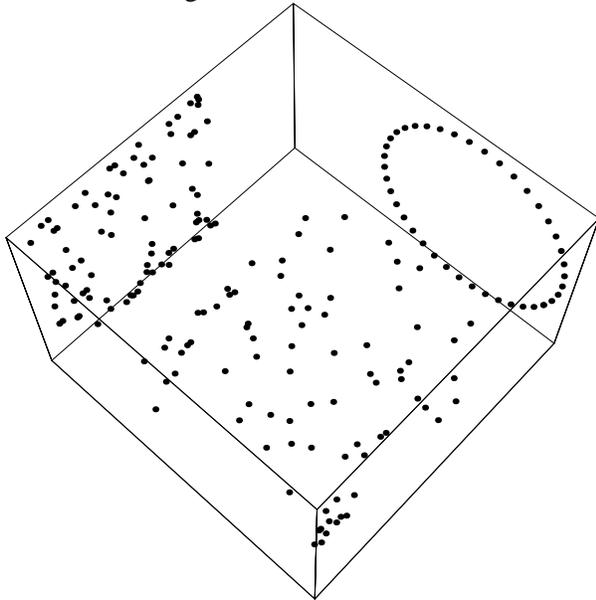


Figure 3: Two projections of 3-D data onto two orthogonal 2-D planes.

Specifically, projection pursuit optimizes (*pursues*) a criterion for a *projection* of a multivariate data set. The criterion of a projection is calculated with an objective function, called the *projection index*, that measures interesting structure within a view [8]. PP, then, augments the projection hoping to make the index more successful. One of several common optimization techniques can drive this augmentation of the projection. When complete, PP reports what projection of the multivariate data set had the most successful index. This information about the successful projection also furnishes the identity of the subset of the dataset’s dimensions (or metrics) that contributed to this projection.

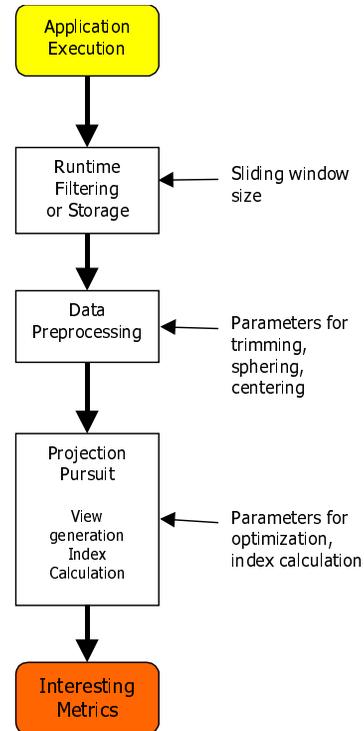


Figure 4: Data flow for Projection Pursuit System.

More formally, the n metrics (as presented in Equation 1) collectively define a vector basis for an n -dimensional coordinate system, the n orthonormal unit vectors. Projection pursuit identifies a smaller set k of orthonormal vectors that are each a linear combination of the original n vectors. The largest components of these projection vectors represent the most successful metrics (i.e., the set of metrics that contributed to a successful projection). Typically, the dimensionality k of the projection is two or three, so one can easily comprehend the resulting metric space. Moreover, because the most important metrics are represented as the largest components of the projection vectors, an algorithm can inspect the magnitude of these components to automatically select the subset of metrics to be recorded. This process of reducing the number of metrics that the system must record ultimately reduces total data volume.

Figure 4 illustrates the flow of data for the projection pursuit algorithm as applied to our performance analysis system. First, instrumentation generates raw performance metrics that can be any combination of physical and logical resource measurements including hardware counters, system software statistics,

library information, and application data. These metrics flow as events to filtering modules or storage. Next, analysis begins with data preprocessing that converts the raw performance data into a suitable metric space for PP. Finally, PP analyzes many views of the data and calculates the corresponding index for each view. The most successful index provides the most interesting metrics.

```

Input smoothed data X
Create Y by sphering, centering, and scaling data X
Calculate product moment tensors on Y
For iterations
    Generate initial projection directions(a,b,c)
    Do
        Modify projection directions(a,b,c) in Y
        Calculate Power sums
        Calculate k-statistics
        Calculate projection index and derivatives
    Until optimal?
Done
Output most successful projection solution

```

Figure 5: Projection Pursuit Algorithm.

Figure 5 outlines our implementation for the projection pursuit algorithm. The input data set X is a matrix with P rows and n columns:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{p1} & x_{p2} & \cdots & x_{pn} \end{bmatrix}$$

where

$$x_{ij} = \frac{1}{W} \sum_{k=0}^{w-1} m_{ij}(t-k) \quad (2)$$

$$i = 1, \dots, P; j = 1, \dots, n$$

That is, each processor records at most n metrics, and these n metrics from P processors serve as input to PP.

3.2 Data Preprocessing

Initially, the smoothed performance metrics in X require preprocessing prior to supplying them to the PP algorithm. We execute several operations on the input data. First, as shown in Equation 2, we create continuous metrics from discrete events with smoothing. Second, we trim the data. Finally, we center and sphere the data, before passing it to

PP.

3.2.1 Smoothing

PP expects a metric space where processor behaviors over time are temporal trajectories through a continuous metric space. In contrast, most monitoring systems, including our event-based systems, define actions at discrete times and locations. To solve this problem of continuous performance trajectories, we smooth metrics for key events. Our smoothing uses a sliding window average technique as shown in Equation 2. The user specifies the length of the sliding window w . Although this sliding window average has the disadvantage of being a non-linear band pass filter that eliminates rapid variations, this technique for smoothing is computationally inexpensive and widely accepted.

3.2.2 Trimming

Outliers tend to influence the projections; so we can modify the data to either remove the outliers altogether, or to reduce their influence on the dataset. Our implementation has several trimming methods. The first method does not alter the data set, however possible outliers are indicated and logged. Second, the implementation can remove outliers from the dataset. The third method uses log trimming to change the outlier's distance from the origin from r to $(1 + \ln r)$. The final method is similar to the third method but the trimming uses a square root function to change the outlier's distance from the origin from r to $3 - (2/\sqrt{r})$.

Our experiments indicate that for our performance metric traces, the third and fourth methods do not produce significantly different results from the first method. Therefore, we do not trim the data.

3.2.3 Centering

Some performance metrics have relatively arbitrary origins, so we *center* each metric to have zero mean. To each element of X , we apply $x'_{ij} = x_{ij} - \bar{x}_i$ across all the values for each metric.

3.2.4 Sphering

Next, *sphering* transforms the metrics so that they have an identity covariance matrix.

Effectively, sphering and centering change the data set so that their answers differ from variance-based techniques, such as principal component analysis. Tukey and Tukey provide additional discussion on centering and sphering in [19]. Our sphering algorithm, taken from [15], applies a linear transformation Q to the centered data X' . The result $Y=QX'$ remains centered and has variance $S_y=QSQ^T$. The convenient choice of $Q=S^{-0.5}$ ensures that S_y is the identity matrix. Q is easily computed from the principal components.

3.3 Projection Pursuit Algorithm

Once the smoothed performance metrics X are transformed into preprocessed data Y , the fundamental portion of the projection pursuit algorithm begins. This algorithm, as given in Figure 5, enters a loop that evaluates dataset Y with several different initial projection directions. Nason [15] provides additional details on this algorithm.

For each initial direction, the algorithm repeats the optimization process. The algorithm first calculates the product moment tensors of the input data Y . These tensors provide computationally efficient methods for all evaluations of the index and its derivatives. Then, the power sums and k-statistics calculations generate a new projection of the data Y using the projection directions (a,b,c) . With each new projection of Y , the algorithm calculates a new index and derivatives. The index represents the *success* of the current projection of Y while the derivatives furnish the proximity to a local maximum and the direction that should be followed to maximize the index. When optimality is found for a particular projection direction, the algorithm saves this direction and continues trying new projection directions for a finite, user-specified number of iterations. When the algorithm finishes trying these different initial projections, it returns the most successful projection in a structure called the *varimax matrix*. The results of the varimax matrix are easily interpreted where the most important three metrics correspond to the maximum value for each column (x, y, z) such that there are three distinct metrics.

PP attempts to find linear combinations of the original variables to maximize the index. In other words [15], these linear combinations can be thought of as projections onto a projection vector

$a: Z=a^T Y$. Let $I(a)$ denote a generic projection index. The optimization problem for PP then becomes

$$\max I(a) \text{ subject to } a^T a=1.$$

Extending the projection index to 3-D, we can change the optimization problem to

$$\begin{aligned} \max I(a,b,c) \text{ subject to} \\ a^T a=1, b^T b=1, c^T c=1 \text{ and} \\ a^T b=0, a^T c=0, b^T c=0. \end{aligned}$$

The additional three constrains (e.g., $b^T c=0$) guarantee that the three projection vectors are orthogonal unit vectors.

In our implementation of PP, the initial projection directions are randomly generated vectors using 128 byte random seeds. This randomization allows the algorithm to quickly generate multiple views and prevents any unnecessary bias towards a particular projection direction.

3.3.1 Projection Index

Clearly, the choice of the projection index dramatically influences the results of PP. The index usually measures some physical property of the data, such as clottedness or the difference between the Shannon entropies of the projected data density and the standard normal density. In one model by Friedman-Tukey [9], the index measures clottedness and the optimization algorithm uses a general hill climbing technique to generate new views of the subspace.

Comparatively, projection pursuit differs from principal component analysis (PCA) because if sample variance is selected as the projection index, then PP creates results similar to PCA (albeit PCA itself provides an analytical solution directly [15]). In fact, other methods including discriminate analysis and factor rotation are special cases of PP.

Our implementation uses the 3-D index described by [15]. Nason extended a one-dimensional index formulated by Jones and Sibson to three dimensions. The original one-dimensional index is based on the approximation of the difference between the Shannon entropies of the projected data density, f , and the standard normal density, ϕ . In the one-dimensional case, the

equation is

$$\int f(x) \log f(x) dx - \int \phi(x) \log \phi(x) dx$$

3.3.2 "Interestingness"

In PP, the index provides an objective measure of *interestingness* for each view of the dataset. Although *interestingness* has a variety of definitions, it is generally considered to be an overall measure of pattern value [7]. This measure can include notions such as novelty, understandability, usefulness, and validity. Unfortunately, notions like understandability are subjective and frequently, difficult to quantify.

Our index for PP, created by Nason, locates non-normal, novel views of the centered, sphered dataset. For this index, we were chiefly interested in a focus on novelty in the data, and we were less interested in these other notions. Also, because of the randomness inherent in the projection vector generation and the optimization process, PP does not guarantee that two similar datasets produce comparable PP results.

3.4 Implementation Issues

3.4.1 Projection Pursuit Updates

As noted earlier, projection pursuit selects interesting metrics from all of the smoothed input data in $M(t)$ at some discrete point in time. Consequently, projection pursuit must be performed periodically on consecutive snapshots of $M(t)$. Like $M(t)$, the results of PP (or the importance of individual metrics for performance analysis) vary with time. Hence, the frequency of triggering projection pursuit becomes an important issue. If the frequency is too high, then the performance analysis system generates additional overhead. On the other hand, if PP is seldom triggered, then the performance analysis system might retain one set of interesting metrics when, in fact, the set of interesting metrics has changed.

3.4.2 Degenerate Performance Metrics

The data preprocessing must manage at least one type of degenerate performance metric: any metric that has zero variance across processors. This attribute of the performance data creates a

matrix X that does not allow the PP algorithm to calculate eigenvectors. Therefore, data preprocessing removes any degenerate metric prior to executing the PP algorithm by repackaging X without the degenerate columns.

3.4.3 Data Reduction

By definition, PP selects the three most interesting performance metrics from a larger set of metrics. Clearly, the degree of data reduction is given by the ratio of three to the total number of metrics: $r = 3/n$. Note that PP selects three metrics regardless of the number of actual interesting metrics. In this regard, the key issue is that PP must discard presumably uninteresting performance metrics, and in some cases, it may be very difficult to reduce the dimensionality of the performance metric space in this way. Experience indicates that PP and other statistical methods often achieve this goal; however, it is not guaranteed.

4 Experimental Evaluation

To explore our hypothesis of using projection pursuit to highlight interesting performance metrics, we have constructed an operational prototype for periodic projection pursuit of performance data. We empirically evaluated our prototype with a synthetic trace and with performance data from several applications.

Recall that the general goal of projection pursuit is data reduction through the reduction of the number of collected performance metrics (or the dimensionality of the performance metric space). To this end, it is important that projection pursuit selects appropriate metrics without losing important information. As illustrated in Figure 5, PP relies on many factors for metric selection including sliding window size, projection index, optimization mechanism, jitter control threshold, scaling technique, sphering technique, and a host of other parameters that configure the numerical analysis and optimization within PP.

Our experimental infrastructure consists of a filter that accepts performance data in Pablo SDDF record format [1]. Each SDDF record contains a processor ID, a timestamp, and numerous metric-value pairs. The filter reads these SDDF records directly from a file or from the Autopilot system [17] at runtime. The filter, then,

periodically emits a SDDF record that describes the relative importance of each performance metric as judged by the PP algorithm.

Unless otherwise noted, we tested all the applications on a cluster of twelve UltraSparc 30-248 systems connected by 100Mbs ethernet.

4.1.1 Performance Metrics

Table 1 lists each performance metric that we measured on our target applications. We captured these measurements with a performance daemon, which was constructed on Autopilot.

Name	Measurement
cpu[idle]	Percentage usage of CPU time as idle
cpu[user]	Percentage usage of CPU time in user mode
cpu[kern]	Percentage usage of CPU time in kernel mode
pswitch	Number of CPU context switches
intr	Number of interrupts
sysfork	Number of system forks
bread	Number of logical disk reads
bwrite	Number of logical disk writes
phread	Number of physical disk reads
phwrite	Number of physical disk writes
packetsin	Number of network packets received
packetsout	Number of network packets sent

Table 1: Performance Metrics

The operating system maintains these measurements as 64-bit monotonically increasing counters. The performance daemon periodically samples these counters and takes the difference from the previous sample. The daemon records this difference, along with other information including a timestamp and location identifier.

4.2 Cactus

First, we traced a distributed application for numerical relativity, called Cactus. The distributed version of Cactus is primarily built on MPI and Fortran 90. The Cactus developers describe Cactus and the underlying physics of numerical relativity in [4]. We ran the application on eight Sun UltraSparc 148 workstations that were connected by a 100Mbs Ethernet.

During application execution, we captured nine performance metrics on each node and these metrics were written to a SDDF trace file. The trace spans the entire run of the application, from 0

to approximately 700 seconds. Figure 6 and Figure 7 provide one portion of these metric traces for processor 0 and processor 6, respectively. Similar traces exist for the other six nodes.

In these figures, each metric has been normalized within its range over the entire trace. Also, to prevent overlaps and highlight data trends, we offset each metric.

We, then, fed this trace file into our PP prototype. We set the sliding window size to 10 seconds and PP update period to 5 seconds. Hence, the metric selection for PP changes in 5 second intervals.

Figure 8 shows the results of the PP prototype for this portion of trace. Three marks indicate which three metrics PP selected during each 5 second time interval.

Over the selected interval, PP focused primarily on metrics `pswitch` and `intr` with a secondary emphasis on the three metrics for the CPU. As represented in Figure 6 and Figure 7, metrics `pswitch` and `intr` appear unrelated across processors, and PP selects these metrics consistently. Meanwhile, during this interval, PP never selects metrics `memfree`, `pgpgin`, `pgswpin`, or `sysfork`. All four of these metrics appear nearly constant across all processors.

4.3 CG-Heat

CG-Heat is a heat diffusion simulation that solves the implicit diffusion PDE using a conjugate gradient solver over each timestep. This code is built with FORTRAN 90 and MPI. Figure 9 and Figure 10 illustrate the traces for the performance metrics on processors 0 and 2, respectively.

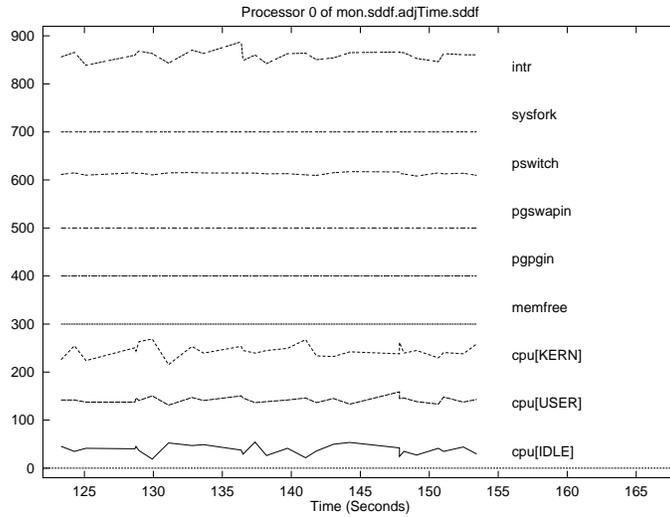


Figure 6: Processor 0 of Cactus.

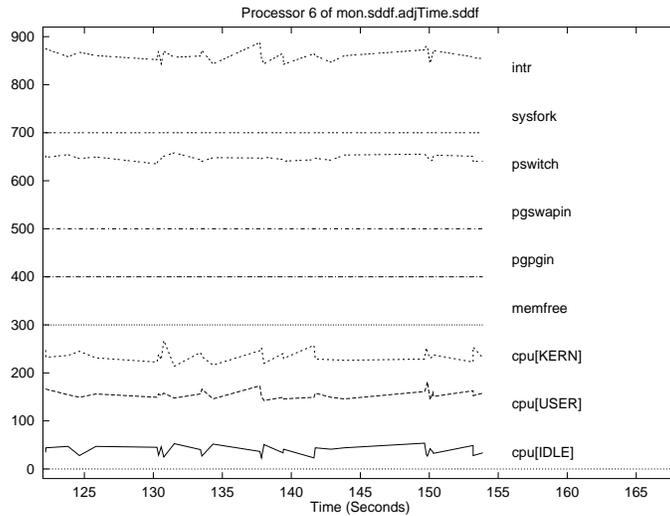


Figure 7: Processor 6 of Cactus.

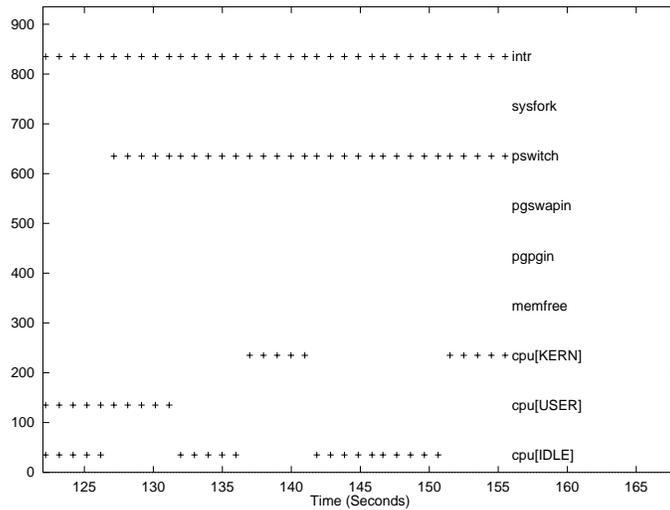


Figure 8: PP result for Cactus.

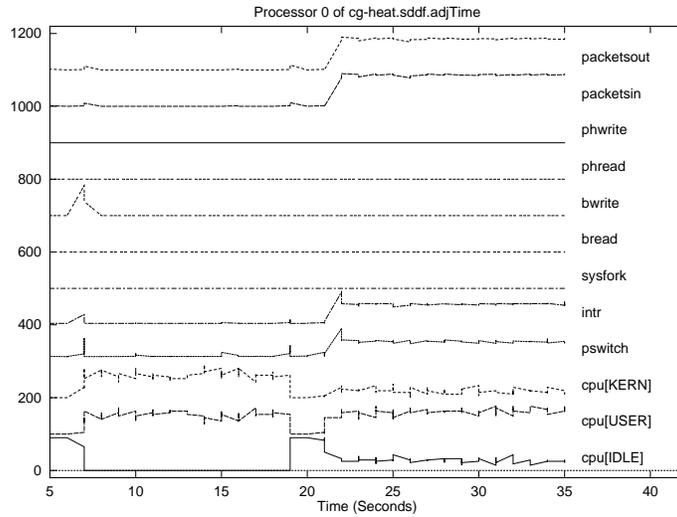


Figure 9: Processor 0 of CG-Heat.

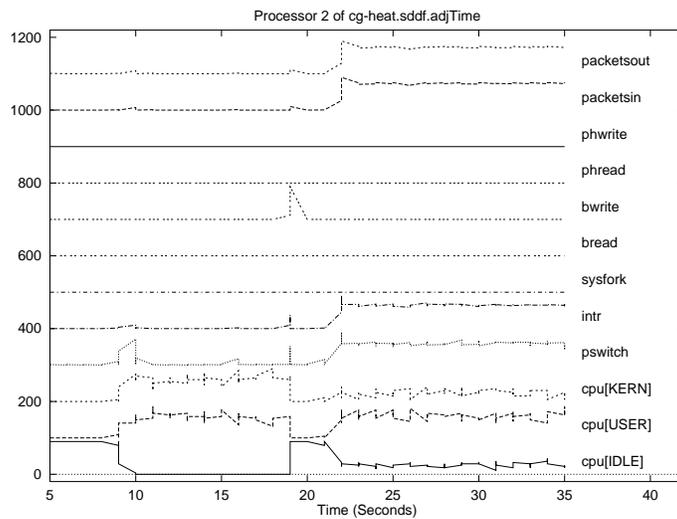


Figure 10: Processor 2 of CG-Heat.

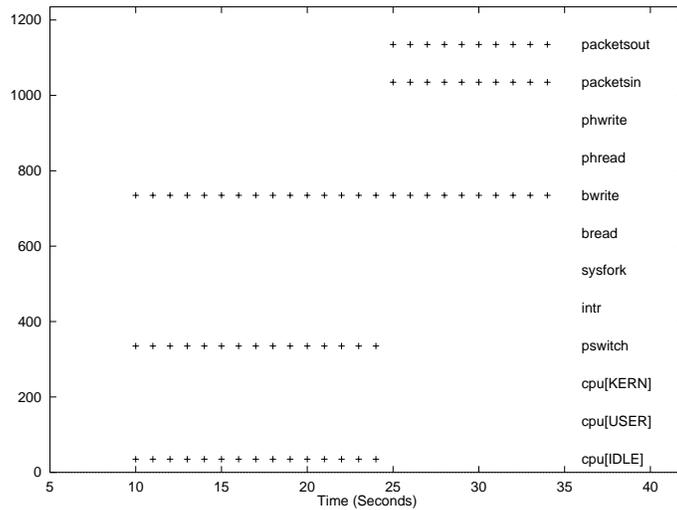


Figure 11: PP result for CG-Heat.

During application execution, we captured twelve performance metrics on each node. The trace spans the entire run of the application, from 0 to approximately 120 seconds. Figure 9 and Figure 10 show a segment of these metric traces for processor 0 and processor 2, respectively. This segment shows the startup section of CG-Heat. Similar traces exist for the other 10 nodes. For our experiment, we set the sliding window size to 10 seconds and PP update period to 5 seconds. Figure 11 shows the results of the PP prototype for this trace segment.

Over the selected interval, PP focused primarily on metrics `bwrite`, `pswitch`, `cpu[idle]`, and `packetsin/out` while ignoring the other metrics. As represented in Figure 9 and Figure 10, metric `bwrite` appears unrelated across these representatives. Further examination of the other processors reinforced the notion that `bwrite` appears random across processors. PP selects `bwrite` consistently.

Although the remaining metrics selected by PP are similar across processors, small differences do exist. For instance, for `packetsin` and `packetsout`, the interval from 20 to 25 seconds, both metrics increased substantially; however, processor 2 had a spike in both metrics while processor 0 had a smoother increase. The remaining ten metrics had trends most similar to processor 0.

For this trace, PP properly ignores the nearly constant metrics during this segment including `phwrite`, `phread`, `bread`, and `sysfork`. In addition, PP ignores metrics that have very similar trends even if they are not constant: `intr`, `cpu[kern]`, and `cpu[user]`.

4.4 Sweep3D

Sweep3D [11] is a solver for the 3-D, time-independent, particle transport equation on an orthogonal mesh. The solver computes along wavefronts in the mesh in eight diagonal directions through the cube. This code is built on FORTRAN and MPI.

As with CG-Heat, during application execution, we captured twelve performance metrics on each node and these metrics were written to a SDDF trace file. The trace spans the entire run of the application, from 0 to approximately 415 seconds. Figure 12 and Figure

13 provide one section of these metric traces for processor 0 and processor 8, respectively. Similar traces exist for the other ten nodes. This Sweep3D trace extends the entire duration of the application execution.

We, then, fed this trace file into our PP prototype. We set the sliding window size to 4 seconds and PP update period to 2 seconds. Hence, the metric selection for PP changes in 2 second intervals, as opposed to the 5 second intervals we used previously.

Over the selected interval, PP primarily focused on four metrics: `pswitch`, `bwrite`, and `packetsin/out` with a distant and secondary emphasis on the two metrics `bread` and `intr`. As represented in Figure 12 and Figure 13, both `packetsin` and `packetsout` of processor 8 have a different periodicity than processor 0. On the other processors, these two metrics appear to have slightly different periodicity as well as shown in Figure 17. PP also selects `bwrite` often because the magnitude of the spikes and their frequency vary considerably across processors as Figure 16 illustrates.

As in the earlier cases, PP never selects metrics `phread`, `phwrite`, `sysfork`, `CPU[kern]`, `CPU[user]`, or `CPU[idle]`. Clearly, `phread`, `phwrite`, and `sysfork` are nearly constant across all processors, so PP never selects them.

Interestingly, PP does not select any of the CPU metrics, even though both `CPU[user]` and `CPU[idle]` exhibit considerable variance across processors as Figure 15 portrays for `CPU[user]`. Note that in a large majority of the traces for `CPU[user]` are at 100% with both periodic drops. Most of these drops approach 0%, especially drops taking more than a second. This fact when combined with smoothing provides a relatively small variance that is usually confined to two or three outliers out of twelve.

Unfortunately, the empty time intervals in the Sweep3D figures identify no metrics because our current implementation of PP suffers occasional numerical instabilities. These instabilities force the PP algorithm to terminate early and, as a result, PP does not output a selection for that interval.

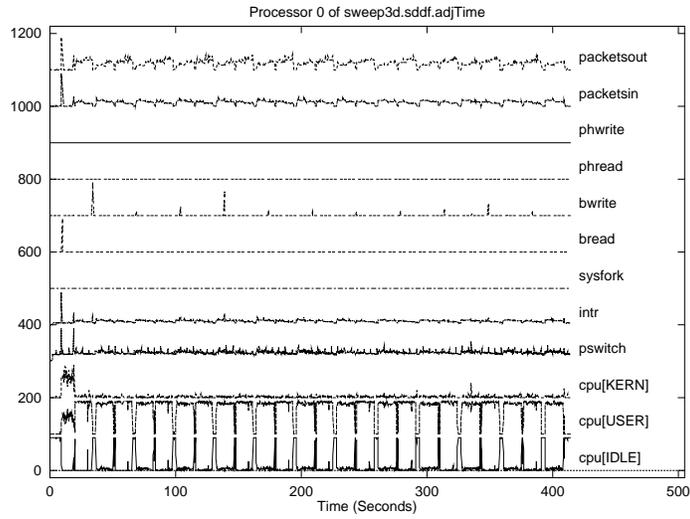


Figure 12: Processor 0 of Sweep 3D.

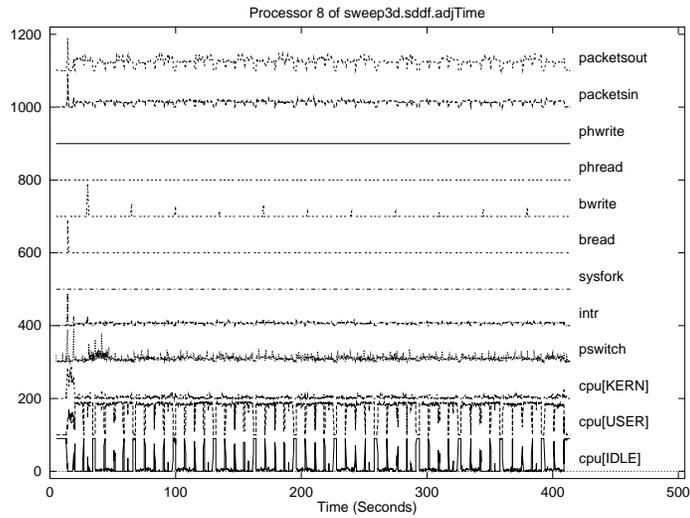


Figure 13: Processor 8 of Sweep 3D.

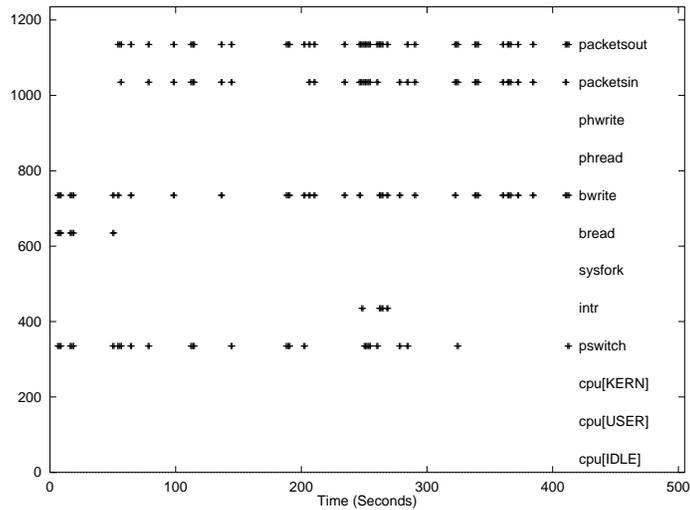


Figure 14: PP result for Sweep 3D.

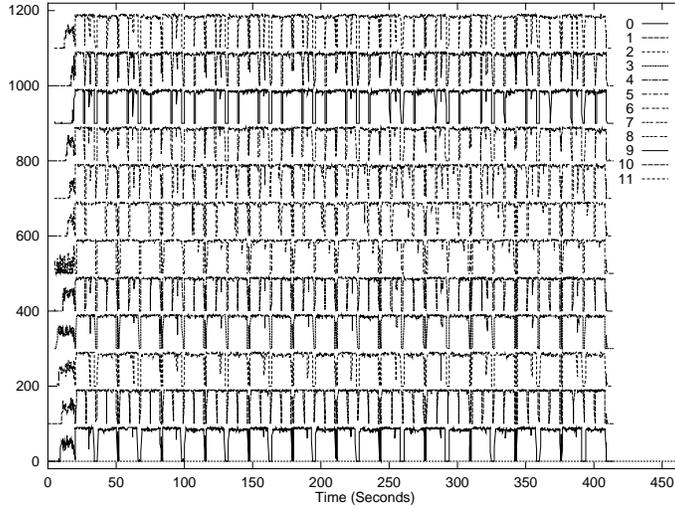


Figure 15: CPU[user] across processors for Sweep3D.

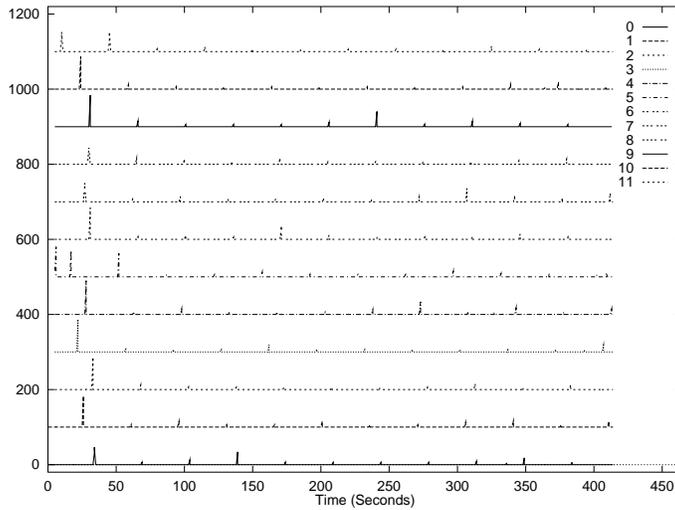


Figure 16: bwrite across processors for Sweep3D.

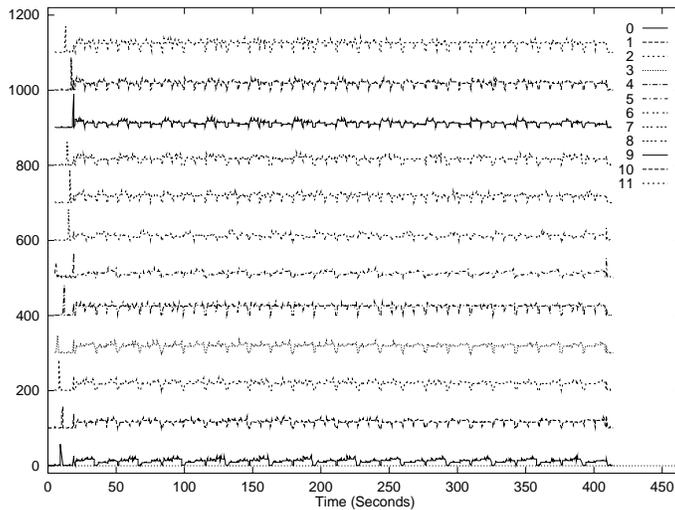


Figure 17: packetsout across processors for Sweep3D.

4.5 SMG98

SMG98 is a parallel semicoarsening multigrid solver for the linear systems arising from finite difference, finite volume, or finite element discretizations of the diffusion equation, $\nabla \cdot (D\nabla u) + \sigma u = f$ on logically rectangular grids. The code solves both 2D and 3D problems with discretization stencils of up to 9-point in 2D and up to 27-point in 3D. [5] furnishes details on the algorithm and its parallel implementation and performance. This code is a distributed memory application based on C and MPI.

During application execution, we captured twelve performance metrics on all workstations at the frequency of 2 Hertz. All metrics were written to a SDDF trace file, spanning from 0 to approximately 120 seconds. Figure 19 and Figure 20 display a portion of the SMG98 traces on processors 2 and 6, respectively. As with the earlier traces, each metric has been normalized within its range over the trace and the metrics are offset to help discriminate trends.

Figure 21 shows the results of PP for this trace segment. During this evaluation, we set the sliding window size to 4 seconds and the PP update interval to 2 seconds. For this segment, PP focused on five metrics: `packetsin`, `packetsout`, `bwrite`, `bread`, and `pswitch`. The three metrics of `packetsin`, `packetsout`, and `pswitch` vary considerably over the trace segment due to the setup phases of SMG98. Although these phases are very closely related given the granularity of our measurements, PP selects them because they vary across all processors. Of the twelve metrics, PP selects `bwrite` and `pswitch` most often. As in the earlier examples, `bwrite` appears totally unrelated to the other metrics as well as the `bwrite` metric on all other processors. As such, PP appropriately selects it. On the other hand, PP chooses `pswitch` during the startup phase of SMG when it appears different across processors, as Figure 19 and Figure 20 show. After this startup phase, `pswitch` becomes similar across processors, resulting in fewer selections by PP.

During this segment, PP never selects `sysfork`, `intr`, `cpu[KERN]`, `cpu[USER]`, `cpu[IDLE]`, `phread`, or `phwrite`. As before, because the metrics `sysfork`, `phread`, and `phwrite` are nearly constant, PP never selects them. On the other hand, `intr`, `cpu[KERN]`,

`cpu[USER]`, and `cpu[IDLE]` are surprisingly similar across all processors. Although these metrics do vary, they are more consistent across processors than the remaining metrics. As we mentioned with the discussion of Sweep3D, smoothing eliminates many of the rapid changes in each metric trace.

Note that the empty time intervals in the SMG98 figures identify no metrics due to the numerical instabilities in our PP implementation.

4.6 Data Reduction

PP reduces the data volume for the monitoring system by periodically selecting a subset of metrics that appears *interesting*. As mentioned earlier, PP selects three metrics from n performance metrics, so the resulting data reduction is approximately $r = 3/n$; however, all records must contain an event identifier, processor number, and a timestamp. Although, this overhead on each record reduces the possible gains for PP, the benefits of using PP grow as the number of metrics grows.

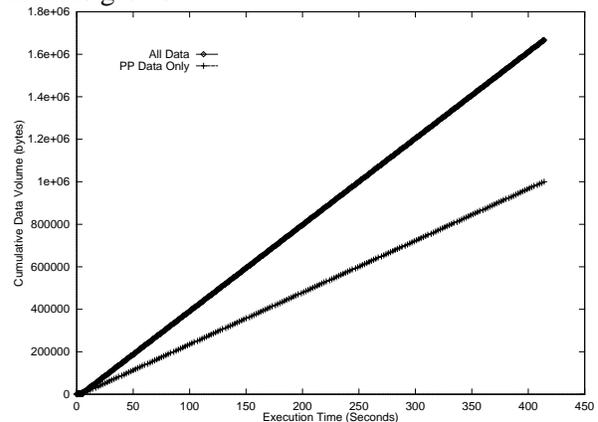


Figure 18: Data reduction due to PP metric selection for Sweep 3D.

For our Sweep 3D example, Figure 18 illustrates the cumulative data reduction due to PP over the run of the entire application if only those metrics selected by PP are captured. For this test on twelve metrics, PP reduces the data volume by 41%. Remember that the advantages of PP increase with the increase in the number of metrics; many HPC systems, such as ASCI-scale platforms, can generate more than 50 performance metrics.

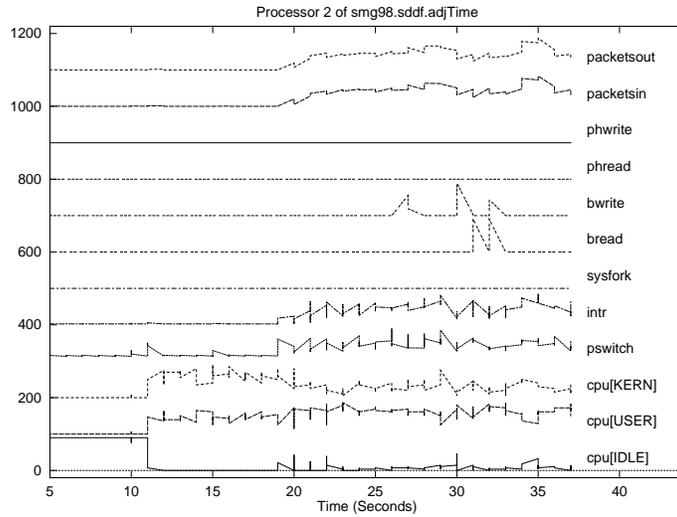


Figure 19: Processor 2 of SMG98.

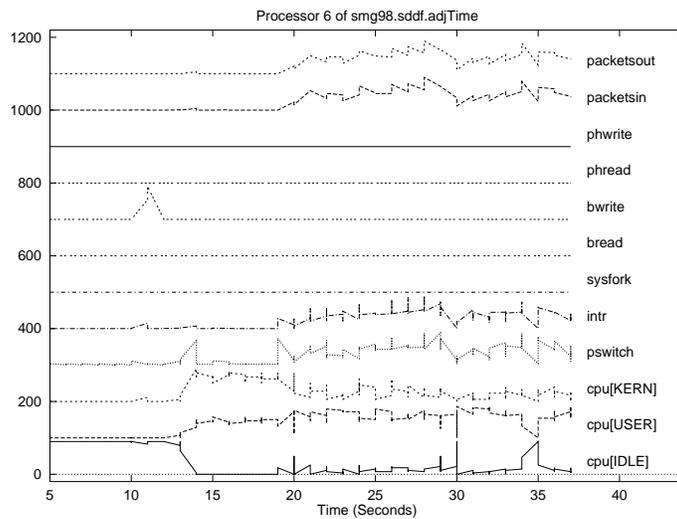


Figure 20: Processor 6 of SMG98.

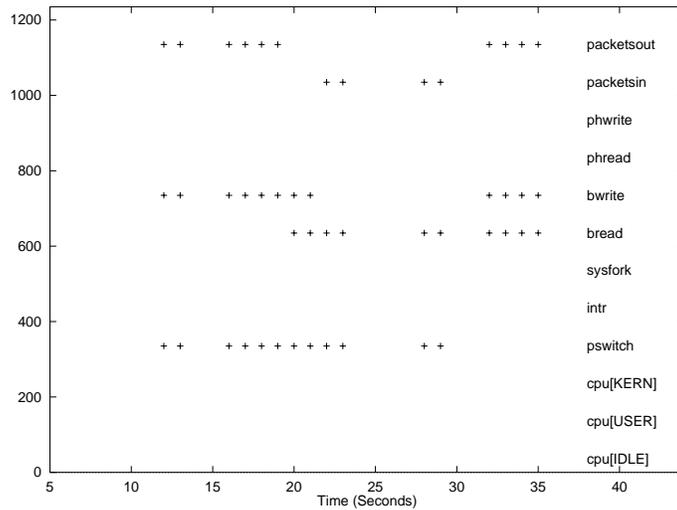


Figure 21: PP result for SMG98.

4.7 Experiment Summary

Several issues reappeared across all of our experiments. First, we found that in our implementation, projection pursuit locates metrics that appear to have little correlation between itself and the same metric on other processors.

Second, the numerical instabilities in our implementation of the PP routines were a significant problem.

Third, PP runtime is proportional to both the number of metrics and the number of processors. We found that our PP filter needed between 0.4 seconds and 1.9 seconds to calculate an answer when $n=12$ and $P=12$.

Fourth, the highest resolution of time in our experiments was seconds. To allow users the capability to capture high frequency changes in their application, we must increase this resolution.

Fifth, smoothing clearly affects the raw data. We need to compare our sliding window average technique to other techniques.

5 Related Work

Various research efforts have produced several strategies to cope with these issues of data management and application perturbation. Data reduction strategies chiefly rely on statistical techniques such as averaging, variance analysis, covariance matrices, clustering [16], filtering [3], and principal component analysis (PCA). Additional data reduction strategies use ideas such as critical path analysis that help the instrumentation system focus on important application components. Strategies for managing perturbation also use these statistical techniques in addition to dynamic instrumentation [14] and minimizing instrumentation requirements [2].

In comparison to this related work, projection pursuit is most closely related to other statistical techniques like correlation analysis and clustering. Correlation or covariance analysis reveals linear associations among input variables; unfortunately, this analysis does not identify non-linear relationships. Also, these descriptive statistics can be very sensitive to outliers and may indicate linear association when little exists [10]. One popular analysis method, principal component analysis (PCA), relies on a measure of sample variance to analyze the data, so it is subject to these constraints as well.

5.1.1 Clustering

Clustering (or segmentation), as described in both [18] and [16] is a well-know data analysis technique that categorizes some raw dataset in the hopes of simplifying the analysis task. Clustering separates data points into clusters where points that belong to the same cluster are more similar than to points in different clusters. When used for performance analysis [16], dynamic clustering identifies clusters of processors with similar performance metric trajectories and then, it selects one processor from each cluster to represent that cluster and to gather detailed performance information. Other members of the cluster decrease their collection rate for performance information. Clustering can result in considerable savings in data volume and instrumentation, especially when many processors have performance metrics that form a few basic equivalence classes, as is the case with SPMD executions.

Although statistical clustering can reduce the number of processors or tasks from which event data must be recorded, it does not reduce the number of metrics or (equivalently) the dimensionality of the metric space. Even after clustering identifies a small number of processor representatives, the total data volume may remain high. As an example, when analyzing the performance of WWW servers [12], researchers found it necessary to capture nearly fifty metrics on request types, processor, network, and memory usage to reliably identify performance problems.

In this regard, clustering identifies clusters of processors with common performance characteristics, but clustering does not provide information on which metrics caused separate clusters. This is precisely what projection pursuit provides: valuable information about *why* processors have different characteristics, and consequently, which metrics possibly deserve more attention during performance analysis.

6 Summary

With a large and growing set of performance metrics in complex, high performance computing systems, performance analysts often have difficulty predicting which performance metrics are important. Dynamic statistical projection pursuit is one technique that can help focus

performance analysis on *interesting* metrics. It identifies metrics that do not have well-understood structure. In this regard, projection pursuit provides a novel addition to a suite of important performance analysis techniques that include clustering and covariance analysis.

With regard to our earlier motivation, projection pursuit (*with our index by Nason*) does, in fact, provide users with additional, novel information about their performance data. This information is especially useful when users want to ask the question “what metrics appear least correlated across the processors within my application?” Such a question is particularly useful for examining SPMD-type applications.

To use projection pursuit as a policy for automatically managing performance instrumentation, we need several improvements to our implementation of the projection pursuit filter. First, we need a better understanding of the numerical instabilities that cause projection pursuit to fail. This understanding could hopefully lead us to additional preprocessing or different numerical technique that would eliminate most failures. Second, we need to improve the performance of our PP implementation. If users want to capture rapidly changing metrics, then PP must operate at an online speed.

Several opportunities exist for future work in this area of performance analysis. Large systems, such as the ASCI Blue Pacific system with 4000 nodes, can generate prohibitive amounts of performance data. Put simply, on these systems, automated tools to help users identify performance problems are a necessity. Projection pursuit serves as one tool that an analyst can use to interrogate these massive, multidimensional performance spaces.

Acknowledgements

This paper has benefited from the detailed comments of our SC99 reviewers and our colleagues in the Pablo group at the University of Illinois at Urbana-Champaign and at Lawrence Livermore National Laboratory. We are also grateful to the DOE ASCI (<http://www.llnl.gov/ASCI>) program for making the ASCI benchmarks available and to the Cactus team (<http://www.cactuscode.org>) for providing us with the Cactus framework.

References

- [1] R. Aydt, “The Pablo Self-Defining Data Format,” University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, IL, USA 1997.
- [2] T. Ball and J. R. Larus, “Efficient path profiling,” Proc. 29th Annual IEEE/ACM Int’l Symp. Microarchitecture (MICRO-29), 1996, pp. 46-57.
- [3] P. C. Bates, “Debugging heterogeneous distributed systems using event-based models of behavior,” *ACM Trans. Computer Systems*, 13(1):1-31, 1995.
- [4] C. Bona, J. Masso, E. Seidel, and P. Walker, “Three Dimensional Numerical Relativity with a Hyperbolic Formulation,” *Paper submitted to Phys. Rev. D*, 1999.
- [5] P. N. Brown, R. D. Falgout, and J. E. Jones, “Semicoarsening multigrid on distributed memory machines,” *SIAM Journal Scientific Computing*.
- [6] D. Cook, A. Buja, J. Cabrera, and C. Hurley, “Grand Tour and Projection Pursuit,” *J. Computational and Graphical Statistics*, 4:155-172, 1995.
- [7] U. M. Fayyad, “Advances in knowledge discovery and data mining,” . Menlo Park, Calif.: AAAI Press : MIT Press, 1996, pp. xiv, 611.
- [8] J. H. Friedman, “Exploratory Projection Pursuit,” *J. American Statistical Association*, 82:249-266, 1987.
- [9] C. Hurley and A. Buja, “Analyzing high-dimensional data with motion graphics,” *SIAM J. Scientific & Statistical Computing*, 11(6):1193-211, 1990.
- [10] R. A. Johnson, *Applied multivariate statistical analysis*. Englewood Cliffs, New Jersey, USA: Prentice-Hall, 1982.
- [11] K. R. Koch, R. S. Baker, and R. E. Alcouffe, “Solution of the First-Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor,” *Trans. Amer. Nuc. Soc.*, 65(198), 1992.
- [12] T. T. Kwan, R. E. McGrath, and D. A. Reed, “NCSA’s World Wide Web Server: Design and Performance,” *IEEE Computer*, 28(11):68--74, 1995.

- [13] A. D. Malony, D. A. Reed, and H. A. G. Wijshoff, "Performance measurement intrusion and perturbation analysis," *IEEE Trans. Parallel and Distributed Systems*, 3(4):433-50, 1992.
- [14] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, "The Paradyn parallel performance measurement tool," *IEEE Computer*, 28(11):37-46, 1995.
- [15] G. Nason, "Three-Dimensional Projection Pursuit," *J. Royal Statistical Society, Series C*, 44:411-430, 1995.
- [16] D. A. Reed, O. Y. Nickolayev, and P. C. Roth, "Real-Time Statistical Clustering and for Event Trace Reduction," *J. Supercomputing Applications and High-Performance Computing*, 11(2):144-159, 1997.
- [17] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed, "Autopilot: adaptive control of distributed applications," Proc. Seventh Int'l Symp. High Performance Distributed Computing (HPDC), 1998.
- [18] H. Spath, *Cluster analysis algorithms*. West Sussex, England: Ellis Horwood Limited, 1980.
- [19] P. A. Tukey and J. W. Tukey, "Preparation; Prechosen Sequences of Views," in *Interpreting multivariate data*, vol. 1, V. Barnett, Ed. New York: John Wiley & Sons, 1981, pp. 189-213.
- [20] E. J. Wegman and L. Qiang, "High dimensional clustering using parallel coordinates and the grand tour," Proc. 28th Symp. Interface of Computing Science and Statistics, 1996.