Models for On-the-Fly Compensation of Measurement Overhead in Parallel Performance Profiling

Allen D. Malony and Sameer S. Shende

Performance Research Laboratory
Department of Computer and Information Science
University of Oregon, Eugene, OR, USA
{malony,sameer}@cs.uoregon.edu

Abstract. Performance profiling generates measurement overhead during parallel program execution. Measurement overhead, in turn, introduces intrusion in a program's runtime performance behavior. Intrusion can be mitigated by controlling instrumentation degree, allowing a trade-off of accuracy for detail. Alternatively, the accuracy in profile results can be improved by reducing the intrusion error due to measurement overhead. Models for compensation of measurement overhead in parallel performance profiling are described. An approach based on rational reconstruction is used to understand properties of compensation solutions for different parallel scenarios. From this analysis, a general algorithm for on-the-fly overhead assessment and compensation is derived.

Keywords: Performance measurement and analysis, parallel computing, profiling, intrusion, overhead compensation.

1 Introduction

In parallel profiling, performance measurements are made during program execution. There is an *overhead* associated with performance measurement since extra code is being executed and hardware resources (processor, memory, network) consumed. When performance overhead affects the program execution, we speak of *performance (measurement) intrusion*. Performance intrusion, no matter how small, can result in *performance perturbation* [7] where the program's measured performance behavior is "different" from its unmeasured performance. Whereas performance perturbation is difficult to assess, performance intrusion can be quantified by different metrics, the most important of which is dilation in program execution time. This type of intrusion is often reported as a percentage slowdown of total execution time, but the intrusion effects themselves will be distributed throughout the profile results.

Any performance profiling technique, be it based on *statistical profiling* methods (e.g., see [4, 14]) or *measured profiling* methods (e.g., see [2, 9]), will encounter measurement overhead and will also have limitations on what performance phenomena can and cannot be observed [7]. Until there is a systematic basis for

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 72–82, 2005.

judging the validity of differing profiling techniques, it is more productive to focus on those challenges that a profiling method faces to improve the accuracy of its measurement. In this regard, we pose the question whether it is possible to compensate for measurement overhead in performance profiling. What we mean by this is to quantify measurement overhead and remove the overhead from profile calculations. (It is important to note we are not suggesting that by doing so we are "correcting" the effects of overhead on intrusion and perturbation.) Because performance overhead occurs in both measured and statistical profiling, overhead compensation is an important topic of study.

In our Euro-Par 2004 paper [8], we presented overhead compensation techniques that were implemented in the TAU performance system [9] and demonstrated with the NAS parallel benchmarks for both flat and callpath profile analysis. While our results showed improvement in NAS profiling accuracy, as measured by the error in total execution time compared to a non-instrumented run, the compensation models were deficient for parallel execution due to their inability to account for interprocess interactions and dependencies. The contribution of this paper is the modeling of performance overhead compensation in parallel profiling and the design of on-the-fly algorithms based on these models that might be implemented in practical profiling tools.

Section §2 briefly describes the basic models from [8] and how they fail. We discuss the issues that arise with overhead interdependency in parallel execution. In Section §3, we follow a strategy to model parallel overhead compensation for message-based parallel programs based on a *rational reconstruction* of compensation solutions for specific parallel case studies. From the rationally reconstructed models, a general on-the-fly algorithm for overhead analysis and compensation is derived. Conclusions and future work are given in Section §4.

2 Basic Models for Overhead Compensation

In our earlier work [8], we developed techniques for quantifying the overhead of performance profile measurements and correcting the profiling results to compensate for the measurement error introduced. This work was done for two types of profiles: flat profiles and profiles of routine calling paths. The techniques were implemented in the TAUprofiling system [9] and demonstrated on the NAS parallel benchmarks. However, the models we developed were based on a local perspective of how measurement overhead impacted the program's execution. Profiling measurements are, typically, performed for each program thread of execution. (Here we use the term "thread" in a general sense. Shared memory threads and distributed memory processes equally apply.) By a local perspective we mean one that only regards the overhead impact on the process (thread) where the profile measurement was made and overhead incurred.

Consider a message passing parallel program composed of multiple processes. Most profiling tools would produce a separate profile for each process, showing how time was spent in its measured events. Because the profile measurements are made locally to a process, it is reasonable, as a first step, to compensate

for measurement overhead in the process-local profiles only. Our original models do just that. They accounted for the measurement overhead generated during TAUprofiling for each program process (thread) and all its measured events, and then removed the overhead from the inclusive and exclusive performance results calculated during online profiling analysis. The compensation algorithm "corrected" the measurement error in the process profiles in the sense that the local overhead was not included in the local profile results.

The models we developed are necessary for compensating measurement intrusion in parallel computations, but they are not sufficient. Depending on the application's parallel execution behavior, it is possible, even likely, that intrusion effects due to measurement overhead seen on different processes will be interdependent. We use the term "intrusion" specifically here to point out that although measurement overhead occurs locally, its intrusion can have non-local effects. As a result, parallel overhead compensation is more complex. In contrast with our past research on performance perturbation analysis [10–12], here we do not want to resort to post-mortem parallel trace analysis. The problem of overhead compensation in parallel profiling using only profile measurements (not tracing) has not been addressed before. Certainly, we can learn from techniques for trace-based perturbation analysis [13], but because we must perform overhead compensation on-the-fly, the utility of these algorithms will be constrained to deterministic parallel execution, for the same reasons discussed in [7, 13].

At a minimum, algorithms for on-the-fly overhead compensation in parallel profiling must utilize a measurement infrastructure that conveys information between processes at runtime. It is important to note this is not required for trace-based perturbation analysis (since the analysis is offline) and it is what makes compensation in profiling a unique problem. Techniques similar to those used in PHOTON [15] and CCIFT [1] to embed overhead information in MPI messages may aid in the development of such measurement infrastructure. However, we first need to understand how local measurement overhead affects global performance intrusion so that we can construct compensation models and use those models to develop online algorithms.

3 Models of Parallel Overhead Compensation

To address the problem of overhead compensation in parallel execution, we must develop models that describe the effect of measurement overhead on execution intrusion. From these models we can gain insight in how the profiling overheads can then be compensated. However, unlike sequential computation, the models must identify and describe aspects of parallel interaction that may cause different intrusion behavior and, thus, lead to different methods for compensation. We know that the methods will involve the communication of information between parallel threads of execution at the time of their interaction. To be more specific, we will consider parallel compensation in message passing computation. The parallel overhead compensation models we present below allow for information about execution delay to be passed between processes during message

communication. The goal is to determine exactly what information needs to be shared and how this information is to be used in compensation analysis. The modeling methodology we develop extends to shared memory parallel computing, but the case for shared memory will not be presented here.

The approach we follow below constructs an understanding of the parallel compensation problem from first principles. We first look at only two processes and then three processes. From this in-depth study, our hope is to gain modeling and analyses understanding that can extend to the general case. We will follow a strategy of *rational reconstruction* where we take scenario measurement cases and reconstruct an "actual" execution as if the measurement overhead were not present. From what we learn, we then derive a model that works for that case and look for consistent properties across the models to formulate a general algorithm for overhead compensation.

The details of overhead removal in the profile calculation are described in our earlier paper [8]. The focus below is on determining the actual overhead value to be removed for each process. These two operations together constitute overhead compensation.

3.1 Two Process Parallel Models

The simplest parallel computation involves only two processes which exchange messages during execution. Measurement-based profiling will introduce overhead and intrusion local to each process that carries between the processes as they interact. To model the intrusion and determine what information must be shared for overhead compensation, we consider the following two-process scenarios:

One send	Process P1 sends one message to process P2
Two sends	P1 sends two messages to P2
Handshake	P1 sends one message to P2, then P2 sends one message to P1
General	General message send and receive

For each scenario, we enumerate all possible cases for overhead relations between the processes (what is called the "measured execution" model) and for each case derive a representation of the execution with the overhead removed (what is called the "approximated execution" model). We determine the overhead-free approximation using a rational reconstruction of the "actual" event timings with the measurement overhead removed.

Both models are presented in diagrammatic form. In additional, we present expressions that relate the overhead, waiting, and timing parameters from the measured execution to those "corrected" parameters in the approximated execution. It is important to keep in mind that the goal is to learn from the rational reconstruction of the approximated execution how profile compensation is to be done in the other scenarios, especially the general case. For space reasons, we consider only the *One Send* and *General* scenarios in this paper.

Scenario: One Send. Consider a single message sent between two processes, P1 and P2. Figure 1 shows the two possible cases, distinguishing which process

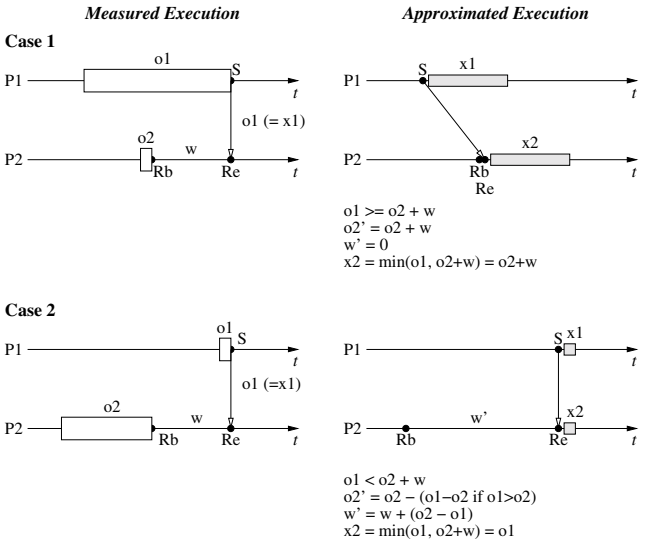

Fig. 1. Two-Process, One-Send – Models and Analysis (Case: 1, 2).

has accumulated more overhead up until the time of the message communication. Execution time advances from left to right and shown on the timelines are send events (S) and receive events (Rb, receive begin; Re, receive end). The overhead on P1 is $o1$ and the overhead on P2 is $o2$. The overhead is shown as a blocked region immediately before the S or Rb events to easily see its size in the figure, but it is actually spread out across the preceding timeline where profiled events occur. Also designated is the waiting time (w) between Rb and Re, assuming waiting time can be measured by the profiling system.

Case 1 occurs when P1's overhead is greater than or equal to P2's overhead plus the waiting time ($o1 \geq o2 + w$). A rational reconstruction of the approximated execution determines that P2 would not have waited for the message (i.e., S would occur earlier than Rb). Hence, the approximated waiting time (designated as w') should be zero, as seen in the approximated execution timeline. Of course, the problem is that P2 has already waited in the measured execution for the message to be received. In order for P2 to know P1's message would have arrived earlier, P1 must communicate this information. Clearly, the information is exactly the value $o1$, P1's overhead. This is indicated in the figure by tagging the message communication arrow with this value.

With P1's overhead information, P2 can determine what to do about the waiting time. The waiting time has already been measured and must be correctly accounted. If the approximated waiting is adjusted to zero, where should the elapsed time represented by w go? If the profiling overhead is to be correctly compensated, the measured waiting time must be attributed to P2's approximated overhead ($o2' = o2 + w$)! This is interesting because it shows how the naive overhead compensation can lead to errors without conveyance of delay

information between sender and receiver. It is also important to note that *Rb* cannot be moved back any further in the approximated execution. This suggests that the only correction we can *ever* make in the receiver is in the waiting time.

The overhead value sent by P1 with the message conveys to P2 the information "this message was delayed being sent by $o1$ amount of time" or "this message would have been sent $o1$ time units earlier." We contend that this is exactly the information needed by P2 to correctly adjust its profiling metrics (i.e., compensate for overhead in parallel execution). We refer to the value sent by P1 as *delay* and will assign the designator x to represent its modeling and analysis that follows. For instance, P1's delay is given by $x1$. In both cases, $x1 = o1$, but it is not always true that delay will be equal to accumulated overhead, as we will see. Now an interesting question arises. How much earlier would future events on process 2 occur in the approximated execution after the message from P1 has been received? In general, each process will maintain a delay value (xi for process Pi) for it to include in its next send message to tell the receiving process how much earlier the message would have been sent. In the approximated execution, for denotational purposes, we show the $x1$ and $x2$ values for P1 and P2 as shaded regions after the last events, S and Re, respectively. We also show an expression for the calculation of $x2$ for this case.

Moving on to the second case, the overhead and waiting time in P2 is greater than what P1 reports (i.e., $o1 < o2 + w$). Rationally, this means that S happens after Rb in the approximated execution. What is the effect on w', the approximated waiting time? It is interesting to see that w' can increase or decrease, depending on the relation of $o1$ to $o2$. (Remember, $o1$ is the same as $x1$ in these cases.) However, the occurrence of Re is certainly dependent on S and, thus, $x2$ will be entirely determined by (and, in fact, equal to) $x1$.

General Scenario. The goal of the two process models is to enumerate the possible cases arising from send/receive message communication. From these cases, we can rationally reconstruct the approximated execution to determine how overhead, waiting, and delay times are to be adjusted. From this reconstruction, we can derive expressions for overhead analysis and correction. The similarity in the case results leads us to propose a general scenario for two processes. This scenario considers an arbitrary message send on one process and corresponding message receive on the other process. Thus, this is a generalization of the *One Send* scenario above. However, we now use the delay values $x1$ and $x2$ instead of the $o1$ and $o2$ overheads in the analysis. The expressions for the two cases are given below (refer to Figure 1):

Case 1	**Case 2**
x1 >= x2 + w	x1 < x2 + w
o2' = o2 + w	o2' = o2 - (x1-x2 if x1>x2)
w' = 0	w' = w + (x2-x1)
x1' = x1	x1' = x1
x2' = min(x1, x2+w) = x2 + w	x2' = min(x1, x2+w) = x1

The importance of the general scenario is the case analysis showing how the delay values are updated and what information is shared between processes during message communication. (Keep in mind that we are arbitrarily designating P1 as the sender and P2 as the receiver. The analysis also applies when P1 is the receiver and P2 the sender, with appropriate reversals of notation in the expressions.) Notice that the overhead values $o1$ (not shown) and $o2$ are accumulated overheads. The $o2$ value is updated here to account for waiting time processing, but whenever any new measurement overhead occurs on P1 or P2, the accumulated overheads $o1$ and $o2$ must be updated accordingly. Similarly, any new measurement overhead must also be added to the delay values $x1$ or $x2$.

Just to be clear, it is the overhead values that are being removed during the profiling calculations. Thus, we want these overhead to be accurately accounted. The conclusion of the two process modeling is that we can handle the parallel overhead compensation for ALL two-process scenarios by applying the general analysis described above on a message-by-message analysis, maintaining the overhead and delay values as the online analysis proceeds.

3.2 Three Process Parallel Models

The question at this point is whether that conclusion applies to three or more processes. That is, can the general two-process analysis be applied on a message-by-message basis to all send/receive messages between any two processes in a multi-process computation and, more importantly, give the desired overhead compensation result? We look at two scenarios with three processes to get a sense of the answer. These scenarios are:

Pipeline	Process P1 sends a message to P2, then P2 sends to P3
Two Receive	Process P1 and P3 sends a message each to process P2

We argue that these two scenarios are enough to elucidate all similar cases regardless of the number of processes. Again, we follow a rational reconstruction approach to determine approximated executions and then derive expressions for updating overhead, waiting time, and delay variables to match the reconstructed executions. Only the *Two Receive* scenarios is described in detail in this paper.

Scenario: Two Receive. When more than two processes are communicating, it is not hard to find a scenario that raises unpleasant issues in our ability to correct overhead intrusion under a different set of receive assumptions. These issues are brought on by the effect of intrusion on message sequencing. The *Two Receive* scenario exposes the problem. Here one process, P2, receives messages from two other processes. There are four cases to consider depending on the relatives sizes of overheads and waiting times. Figures 2 and 3 show two of the cases. For simplicity, we return to looking only at the first messages being sent and received on each process, and consider the initial overheads (not the delays values) in the analysis.

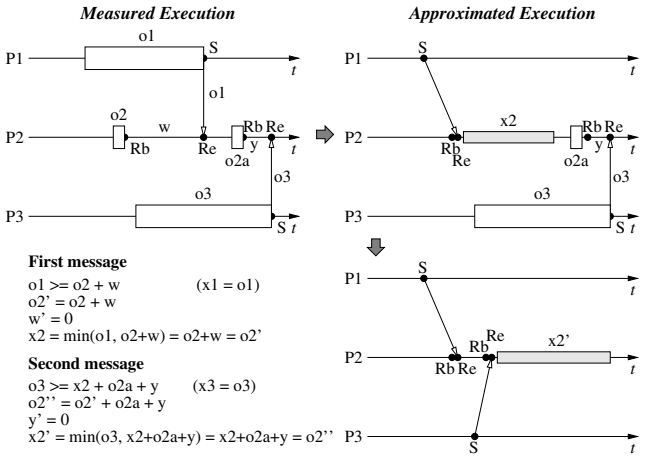

Fig. 2. Three-Process, Two Receive – Models and Analysis (Case 1).

In Figure 2, a two-part approximated execution is shown, with part one (top) giving the state after the first message is processed and part two (bottom) showing the result after the second message is processed. The analysis follows the approach we used before, with new waiting values (w' and y') being calculated and P2's delay value ($x2$) updated. In this case, no waiting time would have occurred, and no adjustment to waiting time is necessary. Otherwise, nothing particularly strange stands out in the approximated result.

What would be a surprising result? If the overhead analysis resulted in a reordering of send events in time, between the measured execution and the approximated execution, then there would be concerns of performance perturbation. In Figure 3, we see the send events changing order in time in the approximated execution, with P3's send taking place before P1's send. As with the other cases, our analysis reflects a message-by-message processing algorithm. In the rational reconstruction, we assume the message communication is explicit and pairs a particular sender and receiver. Under this assumption, the order of messages received by P2 must be maintained in the approximated execution. In this case, is the time reordering of send messages in Figure 3 a problem? In fact, no. It is certainly possible that a process (P2) will first receive a message from a process (P1) sent after another process (P3) sends a message to the receiving process. This just reflects the strict order of P2 receives. However, if we consider receive operations that can match any send, the send reordering exposes a problem with overhead compensation, since the message from P3 should have been received first in the "real" execution.

The application of our overhead compensation models to programs using receive operations that can match any send message results in profile analysis constrained to message orderings *as they are observed* in the measured execution. These message orderings are affected by intrusion and, thus, may not be the

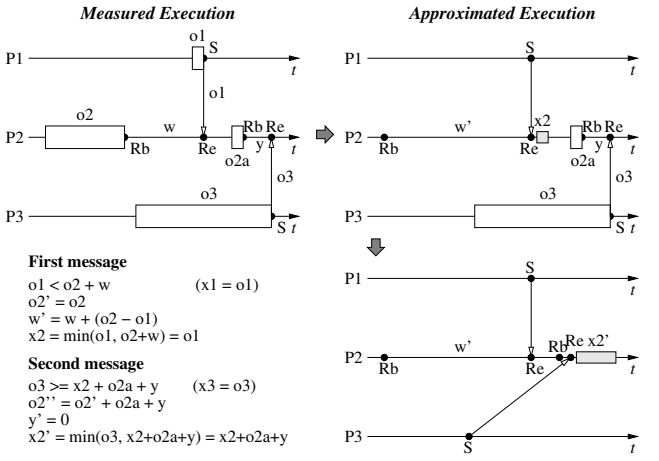

Fig. 3. Three-Process, Two Receive – Models and Analysis (Case: 2).

message orderings that occur in the absence of measurement. However, while it is actually possible to detect reordering occurrences (i.e., measured versus approximated orderings), it is not possible to correct for reordering during online overhead analysis and compensation. Why? There are two reasons. First, our analysis is unable to determine if it is correct to associate a receive event with a different send event. That is, the performance analysis does not know what type of receive is being performed, one that is for a specific sender or one that can accept any sender. Second, even if we know the type of receive operation, it is not possible to know whether changing receive order will affect future receive events. Therefore, the models must, in general, enforce message receive ordering.

3.3 Modeling Summary and General Algorithm

Our above modeling and analysis of measurement overhead in parallel message passing programs has produced three important outcomes. First, the rational reconstructions of the measurement scenarios and the analysis of the approximated executions has resulted in a robust procedure for message-by-message overhead compensation analysis in parallel profiling. It updates correctly waiting times associated with message processing and calculates per process values that capture online the amount a process has been effectively delayed due to measurement overhead and its effects. From this overhead compensation basis, the parallel profiling operations used to update inclusive and exclusive performance can be performed. Second, this analysis requires ALL send messages to be augmented with the delay value of the sender process at the time the message is sent. This information is necessary for the receiving process to apply the analysis procedures. Third, approximation models based on receive type can result in more accurate overhead handling and profile results, but the accuracy gains are anticipated to be minor compared to the processing complexity involved.

We argue that general overhead scenarios for message passing computations can all be addressed from what we learned in the two- and three-process modeling above. A general algorithm for overhead compensation effectively applies the *Two-Process, General* modeling and analysis on a message-by-message basis. The algorithm is composed of three parts:

- Updating of local overhead and delay as a result of local profile measurements.
- Updating of local overhead and delay as a result of messages received and their reported delay.
- Transmission of local delay when a process sends a message.

If the transmission of the delays values can be supported, it should be possible to incorporate this overhead compensation algorithm in a parallel profiling system such as TAU[9].

4 Conclusion and Future Work

Profiling is an important technique for the performance analysis of parallel applications. However, the measurement overhead incurred during profiling can cause intrusions in the parallel performance behavior. Generally speaking, the greater the measurement overhead, the greater the chance the measurement will result in performance intrusion. Thus, there is fundamental tradeoff in profiling methodology concerning the need for measurement detail (as determined by number of events and frequency of occurrence) versus the desired accuracy of profiling results. We argue that without an understanding of how intrusion affects performance behavior and without a way to adjust for intrusion effects in profiling calculations, the accuracy of the profiling results is uncertain. Most parallel profiling tools quantify intrusion as a percentage slowdown in the whole execution and regard this as an implicit measure of profiling goodness. This is unsatisfactory since it assumes overhead is evenly distributed across all threads of execution and all profiling results are uniformly affected.

Our early work in parallel perturbation analysis [11–13] demonstrated the ability to track performance intrusion and remove its effects in performance analysis results. However, there we had the luxury of a fully qualified event trace which included synchronization events that exposed dependent operation. This allowed us to recover execution sequences and derive performance results for an approximated "uninstrumented" execution. While the same perturbation theory applies, when profiling measurements are used, the analysis must be performed online.

This paper contributes models for measurement overhead compensation derived from a rational reconstruction of fundamental parallel profiling scenarios. Using these models we described a general on-the-fly algorithm that can be used for message passing parallel programs. The errors encountered in our earlier work on the NAS parallel benchmarks, resulting from our simpler overhead and compensation models, should now be reduced. However, implementing this algorithms requires the ability to piggyback *delay* values on send messages and

to process the delay values at the receiver. We are currently developing a MPI wrapper library to support delay piggybacking that we can use to validate our approach. Our implementation is intended to be portable to all MPI implementations and will not require transmission of multiple messages. This scheme will be incorporated in the TAU performance system.

References

1. G. Bronevetsky, D. Marques, K. Pingali, and P. Stodghill, "Automated Application-level Checkpointing of MPI Programs," *Principles and Practice of Parallel Programming (PPoPP)*, 2003.
2. L. De Rose, "The Hardware Performance Monitor Toolkit," *Euro-Par Conference*, 2001.
3. A. Fagot and J. de Kergommeaux, "Systems Assessment of the Overhead of Tracing Parallel Programs," *Euromicro Workshop on Parallel and Distributed Processing*, pp. 179–186, 1996.
4. S. Graham, P. Kessler, and M. McKusick, "gprof: A Call Graph Execution Profiler," *SIGPLAN Symposium on Compiler Construction*, pp. 120–126, June 1982.
5. R. Hall, "Call Path Profiling," *International Conference on Software Engineering*, pp. 296–306, 1992.
6. D. Kranzlmüller, R. Reussner, and C. Schaubschläger, "Monitor Overhead Measurement with SKaMPI," *EuroPVM/MPI Conference*, LNCS 1697, pp. 43–50, 1999.
7. A. Malony, "Performance Observability," Ph.D. thesis, University of Illinois, Urbana-Champaign, 1991.
8. A. Malony and S. Shende, "Overhead Compensation in Performance Profiling," *Euro-Par Conference*, LNCS 3149, Springer, pp. 119–132, 2004.
9. A. Malony, et al., "Advances in the TAU Performance System," In V. Getov, M. Gerndt, A. Hoisie, A. Malony, B. Miller (eds.), *Performance Analysis and Grid Computing*, Kluwer, Norwell, MA, pp. 129–144, 2003.
10. A. Malony, D. Reed, and H. Wijshoff, "Performance Measurement Intrusion and Perturbation Analysis," *IEEE Transactions on Parallel and Distributed Systems*, **3**(4):433–450, July 1992.
11. A. Malony and D. Reed, "Models for Performance Perturbation Analysis," *ACM/ONR Workshop on Parallel and Distributed Debugging*, pp. 1–12, May 1991.
12. A. Malony, "Event Based Performance Perturbation: A Case Study," *Principles and Practices of Parallel Programming (PPoPP)*, pp. 201–212, April 1991.
13. S. Sarukkai and A. Malony, "Perturbation Analysis of High-Level Instrumentation for SPMD Programs," *Principles and Practices of Parallel Programming (PPoPP)*, pp. 44–53, May 1993.
14. Unix Programmer's Manual, "prof command," Section 1, Bell Laboratories, Murray Hill, NJ, January 1979.
15. J. Vetter, "Dynamic Statistical Profiling of Communication Activity in Distributed Applications," *ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, ACM, 2002.