

A Framework for Trace Based Performance Analysis of Parallel Programs via Message Passing Graph Traversal

Matthew J. Sottile

Los Alamos National Laboratory, NM

Vaddadi P. Chandu

vchandu@cc.gatech.edu

Georgia Institute of Technology

David A. Bader

Georgia Institute of Technology

Introduction

The ability to understand the factors contributing to parallel program performance are vital for tuning applications for existing platforms and procurement of future platforms based on the performance characteristics of the application set to use the resources. The primary causes of performance issues within distributed memory parallel computers is the latency of the interconnection network and perturbations to applications on processors due to interactions with the operating system and other tasks^[1,3]. One technique for analyzing the performance characteristics of a parallel program is to simulate perturbations in message latency and processor compute time, and propagate these perturbations through subsequent messages and computations to observe their effect on application runtime. We propose a methodology for analyzing the performance characteristics of parallel programs based on message passing traces of their execution on a set of processors. We present this framework in the context of Message Passing Interface library (MPI), but the work itself is not bound to MPI.

What is a Message Passing Graph?

For a parallel program running on p-processing elements, an interconnected p-straight line (or trace) graphs (one trace per processor) can be constructed, with nodes as events and edges as the delay due to communication or operating system

With the order of execution preserved, such a graph can be used to analyze the parallel program, based on the interconnections between the nodes. Such a graph is a Message Passing Graph



Figure 1. A single event in a Message passing graph is illustrated. This figure represents a "send-recv" communication event. "Send" event belongs to one processor and "Recv" event belongs to another. Edges within a processor events represents operating system noise and edges between nodes different processors represent communication latency.

Performance Analysis through Message Passing Graph

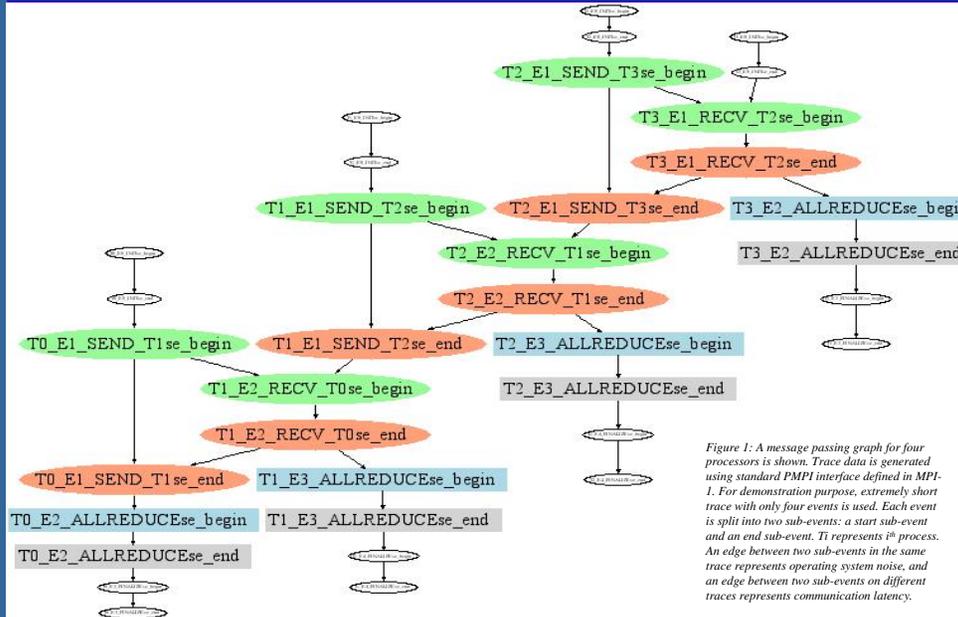


Figure 1: A message passing graph for four processors is shown. Trace data is generated using standard PMPI interface defined in MPI-1. For demonstration purpose, extremely short trace with only four events is used. Each event is split into two sub-events: a start sub-event and an end sub-event. T_i represents i^{th} process. An edge between two sub-events in the same trace represents operating system noise, and an edge between two sub-events on different traces represents communication latency.

- Generate Trace data: Using PMPI interface defined by MPI-1^[4], generate trace data for the parallel program with a time stamp that is provided by high resolution timers
- Gather parameters for target system: Using *microbenchmarks*^[5], probe specific performance parameters that are of interest, for example, communication latency, and operating system noise
- Parameterize Simulated Perturbations: Use the data to build an empirical distribution, such that the shape of the actual distribution is captured
- Inject perturbations: To analyse the application, perturb the performance analyser with the distribution of simulated perturbations obtained in the previous step
- Set Granularity of Analysis: Very fine granularity can be obtained with addition of more sub-events. Sub-events can be added without much effort to the analyser
- Comment: Since this method takes trace data from a real run of the application, the results obtained are expected to be closer to the actual performance of the application in real environment

Preliminary Results on Token Ring^[2]

- n-body simulations, n^2 particles & p-processes
- Each process owns a packet of n/p particles
- P_i sends its packet to $(i+1) \pmod{p}$ th process
- Requires $O(n)$ steps to complete
- Uses Blocking Send-Recv MPI-Primitives

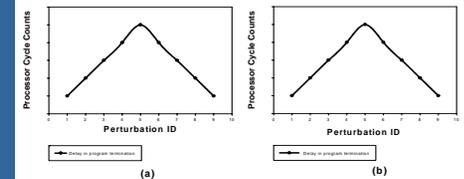


Figure 2: This figure demonstrates the performance analysis of Token Ring implementation. In (a) perturbation in OS Noise is injected and in (b) perturbation in Communication Latency is injected. Token Ring implementation uses only blocking MPI-Primitives, hence, the delay in program termination should follow the nature of perturbations. Intentionally, we injected perturbations of the nature of Gaussian distribution, and as expected the output follows the perturbation. Since we are interested in analysing the working of the performance analyser, values on y-axis are not very important, hence they are not shown.

Conclusions and Future Work

- We present a methodology and prototype of a performance analysis tool that is closer to the performance of application on real machine.
- No changes in the order of execution ensures correctness of the model.
- This preliminary work establishes the feasibility and correctness of the method.
- Future work will add a complete set of MPI-Primitives and a richer set of parameters.

References

- [1] Raj. Jain. The Art of Computer Systems Performance Analysis. John Wiley and Sons, 1991.
- [2] Joseph JaJa. An Introduction to Parallel Algorithms. Addison Wesley, 1992.
- [3] Averil M. Law and W. David Kelton. Simulation Modeling and Analysis. McGraw-Hill, second edition, 1991.
- [4] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-84-230, University of Tennessee, 1984.
- [5] Matthew Sottile and Ronald Minnich. Analysis of Microbenchmarks for the Performance Tuning of Clusters. In Proceedings of Cluster 2004, 2004.