AIM

Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910

301/587-8202

Centimeter

Inches

MANUFACTURED TO AIM STANDARDS
BY APPLIED IMAGE, INC.

Performance Measurement and Analysis Techniques for Parallel and Distributed Programs

Technical Progress Report

(Department of Energy Grant DE-FG02-93ER25176)

August 1, 1993 – July 31, 1994

Principle Investigators:

Barton P. Miller

James R. Larus

Computer Sciences Department

University of Wisconsin

Madison, WI 53706

608-263-3378

bart@cs.wisc.edu

1. SUMMARY OF TECHNICAL PROGRESS

During the first of our grant, we have made substantial progress on our performance tools research. We have concentrated on our techniques for dynamic program instrumentation and on support of high-level parallel languages and environments. In addition, we have performed an interesting study in a related area, virtual memory scheduling for parallel systems.

1.1. Dynamic Instrumentation

Efficient data collection is a critical problem for any system that monitors the performance of a parallel or distributed application. We have estimated that monitoring programs at a reasonable level of detail on current RISC processors can easily generate two megabytes per second per processor of performance data. For a massively parallel computer (say 1000 nodes), this amount of data is impractical to collect for all but the shortest programs. However, to understand the performance of parallel programs, it is necessary to collect data for full-sized data sets running on large numbers of processors. We have developed a new approach to performance instrumentation that defers instrumenting the program until it is in execution, permitting dynamic insertion and alteration of the instrumentation during program execution [3,6].

We defer inserting any instrumentation until the application is in execution. We insert or change instrumentation at any time during execution by modifying the application's binary image. Only the instrumentation required for the currently selected analysis or visualization is inserted. As a result, our technique can collect several orders of magnitude less data than traditional data collection approaches.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

87B

✓

Dynamic instrumentation is now working for the TMC CM-5 and the PVM message passing environment (on Sun workstations). We extended the operating system on the CM-5 to support our instrumentation more efficiently; TMC has incorporated these changes in their release 7.3 of the CMOST operating system. We also worked with Jack Dongarra (ORNL/Univ. of Tenn.) to design extensions to the PVM software to better support development tools [1]. These extensions have been incorporated into release 3.3 of PVM.

1.2. High-Level Parallel Languages

We are also considering programs written in high-level parallel languages. High-level parallel programming languages promise to make programmers' lives easier. They offer portable, conceptually compact notations for specifying parallel programs, and their compilers automatically map programs onto complex parallel machines, freeing programmers from the difficult, error-prone, and sometimes ineffective task of specifying parallel computations explicitly. Unfortunately, effective performance measurement tools for high-level parallel programming languages are difficult to build because they must account for implicit low-level activities created by compilers and must present performance information about those activities in terms of the source code language.

We have developed a new model, NV [5], for the explanation of performance measurements of high-level parallel applications. With NV, we organize a high-level parallel application into a set of abstraction layers that corresponds to the set of layers of software on which the application is built. Each layer of abstraction consists of a collection of nouns (representing code and data elements) and verbs (runtime actions). In NV, mapping functions relate nouns and verbs of a given layer of abstraction to nouns and verbs of other layers of abstraction. Mapping functions preserve the relationships between low-level system-dependent activities that are performed on behalf of high-level system-independent code constructs. These mapping functions allow us to get performance information not easily obtained from current tools.

Using NV as a guide, we have designed and implemented ParaMap, a prototype performance tool for CM Fortran programs running on CM-5 systems. ParaMap has allowed us to study large and long running applications and significantly improve their execution times (73% in one case). ParaMap summarizes system performance at the source code level in terms of CM Fortran arrays, array subsections, and statements. However, if performance problems cannot be fully understood at the source code level, ParaMap allows the user to peel back layers of abstraction to view the runtime system and processor activity while retaining mappings to the appropriate CM Fortran constructs.

Data parallel programs are sensitive to the distribution of data across processor nodes. We are developing techniques that record the inter-node communication caused by the movement of array data between nodes during execution. We formulate a representation of the data distribution and movement in a colored graph. We have developed a simple algorithm that optimizes the coloring of this graph to describe new data distributions that would result in less inter-node communication. From the distribution information, we can write compiler pragmas to be used in the application program.

Using these techniques, we traced the execution of a real data-parallel application (written in CM-Fortran) and collected the array access information. We were able to generate graph colorings that would improve the

application performance (though, for our test application, compiler "optimizations" counter-acted any data-redistribution we specified). The techniques form the basis for future work to monitor array access patterns and generate on-the-fly redistributions of arrays.

We have also been investigating memory systems to support high-level parallel languages. Many high-level parallel programming languages offer desirable semantics that are difficult to implement efficiently on current machines. We have demonstrated that flexible, program-controlled memory systems can help compilers achieve good performance for language constructs that previously appeared too costly to be practical.

Our memory system is called Loosely Coherent Memory (LCM) [2]. LCM is an example of a larger class of memory models, called Reconciliation-Based Shared Memory, that generalize the replication and merge policies of cache-coherent shared-memory. LCM provides a program with dynamic control over memory-system policies, which a compiler can use to implement a language's semantics, improve performance, or detect errors.

We are currently investigating LCM implementations on a variety of platforms that support fine-grained access control. These platforms allow for efficient LCM implementations, and permit exploration of techniques that combine fine-grained access control with LCM's merging policies. One promising synthesis uses access control to record modifications to shared data and LCM to detect potential conflicts, thus performing on-the-fly data race detection.

1.3. Virtual Memory for Parallel Systems

While demand paging is frequently found on workstations, it has rarely been used on larger parallel computers. Operating systems on newer massively parallel processors and networks of workstations, however, support such paging facilities. We have measured the performance of several shared-memory parallel application programs on a system that simulated demand-paged virtual memory. We first experimented with CPU scheduling algorithms and concluded (not surprisingly) that gang-scheduling was a necessity. We then examined a range of possible CPU scheduling policies for when processes incur page faults. For current architectures, we found that a global context switch to a new parallel process is too expensive in relation to the cost of servicing a page fault. A better option is for the faulting node to spin or temporarily execute a local sequential process.

Given a reasonable set of scheduling policies, the parallel application programs that we measured still demonstrated poor performance when running with demand paging. We studied the page reference patterns of these application programs and found that they exhibited little locality of reference. Paging replacement algorithms that predict future behavior based on the recent past do not work well on the programs that we measured.

These programs need most, if not all, of their pages in memory to execute efficiently.

We conclude that demand paging, given the performance characteristics of current parallel systems and networks of workstations, is of limited use. Changes in various system performance characteristics, however, will drive new approaches and virtual memory policies.

2. LIST OF PUBLICATIONS and REPORTS

- (1) B.P. Miller, J.K. Hollingsworth, and M.D. Callaghan, "The Paradyn Parallel Performance Tools and PVM", in **Environments and Tools for Parallel Scientific Computing**, SIAM, J. Dongarra and B. Tourancheua, eds., 1994.
- (2) J.R. Larus, B. Richards, and G. Viswanathan, "LCM: Memory System Support for Parallel Language Implementation", 6th International Conf. on Architectural Support for Programming Languages and Operating Systems, Santa Clara, Calif., October 1994.
- (3) J. Hollingsworth, J.E. Lumpp, and B.P. Miller, "Techniques for Performance Measurement of Parallel Programs", in **Parallel Computers: Theory and Practice**, IEEE Computer Society Press, P. Tvrđik, F. Plasil, and T.L. Casavant, eds., 1994.
- (4) D.C. Burger, R.S. Hyder, B.P. Miller, and D.A. Wood, "Paging Tradeoffs in Distributed Shared-Memory Multiprocessors", *Supercomputing '94*, Washington, DC, November 1994.
- (5) R.B. Irvin and B.P. Miller, "A Performance Tool for High-Level Parallel Programming Languages", *IFIP WG10.3 Conference on Programming Environments for Massively Parallel Distributed Systems*, Ascona, Switzerland, April, 1994.
- (6) J.K. Hollingsworth, B.P. Miller and J. Cargille, "Dynamic Program Instrumentation for Scalable Performance Tools", *Scalable High Performance Computing Conference*, Knoxville, May 1994.
- (7) K. Kunchithapadam and B.P. Miller, "Optimizing Array Distributions in Data-Parallel Programs", *7th Annual Workshop on Languages and Compilers for Parallel Computing*, Ithaca, August 1994.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DATE
FILMED

9/9/94

END