

Performance Analysis Tools

Shirley Moore
shirley@cs.utk.edu
 April 3, 2002

Spring 2002

CS594

1

Optimizing Compiler

- Goal is efficient translation of higher-level language into the fastest possible machine language that accurately represents the high-level language source
- Optimization levels and options
 - No optimization
 - Basic optimization
 - Interprocedural analysis
 - Runtime profile analysis
 - Automatic parallelization

Spring 2002

CS594

2

Classical Optimizations

- Copy propagation
- Constant folding
- Dead code removal
- Strength reduction
- Variable renaming
- Common subexpression elimination
- Loop-invariant code motion
- Induction variable simplification

Spring 2002

CS594

3

Helping the Compiler

- Use straightforward coding style so as not to confuse the compiler
- Look at compiler listing to see where code may need to be restructured
 - Restructure loops
 - Simplify or rearrange memory references

Spring 2002

CS594

4

Timing and Profiling

- Timing an entire program
 - UNIX *time* command outputs
 - User time
 - System time
 - Elapsed time
 - User time + system time = CPU time
 - Additional *time* output
 - Percent utilization
 - Average memory utilization
 - Blocked I/O operations
 - Page faults and swaps

Spring 2002

CS594

5

Timing a Portion of a Program

1. Record the time before you start doing X
2. Do X
3. Record the time at completion of X
4. Subtract the start time from the completion time

Spring 2002

CS594

6

Subroutine Profiling

- Most compilers provide a facility to automatically insert timing calls into your programs at the entry and exit of each routine at compile time or to enable sampling.
- A separate utility (e.g., *prof*, *gprof*) produces a report showing the percentage of time spent in each routine.
- Many performance analysis tools also provide this capability.

Spring 2002

CS594

7

Types of Profiling

- Time-based
- Based on other metrics such as
 - Operation counts
 - Cache and memory event counts

Spring 2002

CS594

8

Possible Profiles

- Sharp profile
 - Most of the time spent in one or two procedures
 - A minor optimization in a heavily executed line of code can have a great effect on overall runtime
 - Typical of engineering and scientific codes built around matrix solutions
- Flat profile
 - Runtime spread fairly evenly across many routines

Spring 2002

CS594

9

Basic Block Profilers

- A basic block is a section of code with only one entrance and one exit.
- If you know how many times the basic block was entered, you know how many times each of the statements in the block was executed, which gives you a line-by-line profile.

Spring 2002

CS594

10

Virtual Memory

- The virtual memory system can slow your program down if it is too large to fit in physical memory or is competing with other jobs for memory resources.
- Use *vmstat* (or similar utility) to monitor paging activity.

Spring 2002

CS594

11

Trace-based Tools

- Collect timestamped trace records during runtime
- Often implemented using an MPI profiling library
- Display timeline view and/or statistical analysis of program execution

Spring 2002

CS594

12

Performance Analysis Tools

- DEEP/MPI
- MPE logging/Jumpshot
- Pablo Performance Analysis Tools
- Paradyn
- Scalea
- TAU
- Vampir
- VProf

Spring 2002

CS594

13

DEEP/MPI

- http://www.psrvc.com/deep_mpi_top.html
- Compile MPI program with *mpiprof* compile driver
- Execute program
- View performance data using the DEEP/MPI interface
- Support the PAPI interface to hardware counters
- Can be used to analyze mixed MPI and shared memory (e.g., OpenMP) programs

Spring 2002

CS594

14

MPE Logging/Jumpshot

- Distributed with MPICH
- <http://www-unix.mcs.anl.gov/mpi/mpich>
- Developed for use with MPICH but can be used with other MPI implementations
- Two log file formats
 - CLOG
 - SLOG
- Timeline view

Spring 2002

CS594

15

Pablo Performance Analysis Tools

- <http://www-pablo.cs.uiuc.edu>
- Base library plus extensions for
 - I/O
 - MPI
 - MPI I/O
- Generates trace records in SDDF (Self Defining Data Format)
- SvPablo
 - Graphical user interface for instrumenting source code and browsing runtime performance data

Spring 2002

CS594

16

Paradyn

- <http://www.cs.wisc.edu/paradyn/>
- Dynamically inserts instrumentation into a running application and analyzes and displays performance information in real-time
- Performance Consultant attempts to automatically identify performance problems

Spring 2002

CS594

17

Scalea

- <http://www.par.univie.ac.at/project/scalea/>
- Scalea Instrumentation System (SIS)
- SISPROFILING Library
- Analysis and Visualization System
 - Profile/Trace analysis
 - Overhead analysis
 - Multiple experiments analysis

Spring 2002

CS594

18

TAU

- Tuning and Analysis Utilities
- <http://www.acl.lanl.gov/tau/>
- Portable profiling and tracing toolkit
- Supports MPI, OpenMP, and mixed MPI/OpenMP

Spring 2002

CS594

19

Vampir

- <http://www.pallas.de/pages/vampir.htm>
- Commercial MPI performance analysis tool from Pallas
- Vampirtrace MPI profiling library + API
- Vampir analysis GUI
 - Timeline view
 - Statistics views

Spring 2002

CS594

20

VProf

- <http://aros.ca.sandia.gov/~cljanss/perf/vprof/>
- Stands for "Visual Profiler"
- Routines to collect statistical profiling information
- Programs to view execution profiles
 - vprof
 - cprof
- <http://aros.ca.sandia.gov/~cljanss/perf/vprof/doc/README.html>

Spring 2002

CS594

21

Tool Evaluation Criteria

- General
 - Ease of use
 - Functionality
 - Robustness
 - Scalability
 - Portability
 - Versatility
- Specific
 - Support for hybrid environments
 - Support for distributed heterogeneous environments
 - Analysis of MPI-2 I/O

Spring 2002

CS594

22