# Performance Analysis and Visualization

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

# Performance Tools

- Empirical data required for design and implementation of parallel programs.
  - Calibrate and validate performance models and implementations.
- How to collect and analyze data?
  - Is a stopwatch sufficient?
- Various supporting tools exist.
  1. Data collection.
  2. Data transformation.
  3. Data visualization.

# Performance Analysis

## 1. Data collection.

- Process by which data about program performance are obtained from an executing program.
- Profiles: time spent in different program parts.
- Counters: event frequencies or cumulative times.
- Event traces: occurences of various events.

## 2. Data transformation.

- Bring data into form to yield insights.
- Reduce data volume.
- Mean values, higher-order statistics, extract profile/counter data from traces.

## 3. Data visualization.

- Performance data are *multidimensional.*
- Execution times communication costs, ...
- Different processor numbers, problem sizes.
- Explore data by bringing them into visual form.

# Performance Tools

## Various Issues.

- Accuracy.
  - Sampling techniques yield less accurate results than counters or timers (clock accuracies!).
- Simplicity.
  - Automatic data collection (little or no programmer intervention).
- Flexibility.
  - Extension to collect additional data or provide different views.
- Intrusiveness.
  - Data collection may affect program behavior.
- Abstraction.
  - Examination of data at a level of abstraction appropriate for programming model.

# Profiles

- Display amount of time spent in different program components.
- Application of sampling techniques.
  - Simple but not necessarily highly accurate.
  - Value of program counter is determined at fixed intervals.
  - Histogram of execution frequencies.
- Advantages:
  - Automatic generation at low costs.
  - Small amount of data.
  - High-level view of program behavior.
- Limitations:
  - No *temporal* aspects of execution.

*First technique for seeking to understand parallel program performance.*

# Counters

- Count or summarize specified events.
  - Number of procedure calls, number of messages, volume of messages, . . . .
- Generation:
  - Compiler-generated code.
  - Code in communication libraries.
  - User-inserted calls.
- Information not be obtained by sampling.
- Variant: *interval timer*.
  - Length of time spent executing particular pieces of code.
  - Accurate determination of time spent in program component.
- Disadvantage: code modifications.

# Traces

- Generate log of time-stamped events.
  - Type, procedure name, destination task.
- Broad study of program behavior:
  - Examine communication relationships.
  - Localize sources of idle times.
  - Identify temporary hot spots.
- Postprocessing to obtain profile, count, interval timer information.
- Generation of higher-order statistics and summary data.
- Disadvantage: huge volume of data.
  - Thousands of megabytes for large program runs.
  - *Probe effect*: measuring changes behavior.
  - Extraction of relevant information difficult.

*See Foster, Table 9.1 and Figure 9.1.*

## Summary of Data Collection

The more detailed, the more intrusive and the more processing effort.

1. Use profile and count information to obtain parameter values needed to complete performance models.
2. Measure execution times for a range of processor counts and problem sizes, and compare results with predictions.
3. If observed and modeled performance differ, use profile and count information to verify basic assumptions of model.
   - Message counts and volumes, load imbalances, replicated computation).
4. If aspects are still unexplained, use tracing and study performance on a few processors. Increase processor number as understanding improves.

# Profile and Counts

## Data Transformation and Visualization

- Typical information:
  - Time spent in each procedure on each processor.
  - Number of times each procedure is called.
  - Number of messages generated.
- Data reduction techniques to reduce number of dimensions:
  - Scalar data: single numbers (total computation time, total message number, mean message size, ...).
  - 1D data: histograms.
  - 2D data: color and a 2D-matrix.

*See Foster, Plates 7 and 8.*

# Traces

- Reduce to one, two, three dimensions and display using histogram techniques.
- *Gantt chart*
  - Horizontal bar chart where each bar represents processor satus as a function of time (Foster, Plates 8 and 12).
  - Status (computing, communicating, idling).
  - Highlight dependencies between program components.
  - Consistent event timing amoung processors!
- *Space-Time Diagrams*
  - Gantt charts augmented by lines connecting send/receive events on different processors (Foster, Plate 8).
  - Identify specific events for which processor is waiting.

# Higher-level Languages

- Data-parallel models (HPF High Performance Fortran).
- Functional models (PACLIB, STURM).
- Semantic gap between source and executed code!
- Small modifications may result in dramatic behavior changes.
- Low-level information of limited value.
  - Source: functional model.
  - Executable: message passing model.

*Information in terms of high-level programming model required!*

# Tools

- Public-domain and commercial:
  - Paragraph (Oak Ridge National Lab).
  - Upshot (Argonne National Lab).
  - Pablo (University of Illinois).
- Custom tools:
  - Data manipulation: awk, perl.
  - Statistical packages: Mathematica, Matlab.
  - Visualization: gnuplot, AVS.

*Simple tools often provide tremendous insights!*