Performance Analysis of HPC Code

**SvPablo**: A Multi-Language Architecture-Independent Performance Analysis System

The **Paradyn** Parallel Performance Measurement Tool

Presented by Nico Zazworka
Motivation (1)

• Understand the performance bottlenecks of my HPC application
  – *Where* they are
  – Of *what* nature they are
  – *When* they occur
Motivation (2)

• Samples from Game of Life:
  – For x=4 processors y=10000 iterations
    z=100x100 field size
    • Which part of the program takes the longest time?
    • Why does it take so long?
  – How well does my program scale?
    – More processors
    – Larger problem space
    – More iterations
  – How does each variable influence the execution time?
    • Which one is the most important?
    • Anomalies: Spikes? Super Linear Speedup?

• To answer these questions we have to measure performance.
Performance Analysis Refinement Cycle

- Instrument
- Modify
- Measure
- Interpret
How did you instrument and measure your own code?

• Insert statements to retrieve wall time
  – At start and end of program
  – At points of interest
• Print times in the end
• Have a closer look at suspicious times
SvPablo vs. Paradyn

• **Where** is the performance problem?

• **Where** is the performance problem?

• **Why** is the application performing poorly?

• **When (in time)** does the problem occur?

• And all this automatically!
SvPablo – overview (1)

- **Source view Pablo**
- **Analysis and visualization**
  - Provides functionality of:
    - **Instrumenting** code
    - **Capturing** performance information
    - **Visualizing** captured metrics
  - Works also for sequential code
SvPablo – overview (2)

• Language independent
  – Can be extended to new contexts (languages and architectures)
  – Currently supported:
    • C, Fortran (77+90), HPF
    • IBM BlueGene/L, Linux Redhat 3.2.3-42, Cray X1E, SGI Altix

• Hardware and software performance data
  – Are summarized during program execution
  – And stored in a language transparent and portable format (SDDF)

• Initial release 1997
• Last release 2006
Paradyn – overview (1)

• Key features:
  – Dynamic insertion of instrumentation calls during runtime
  – $W^3$ search model
    • Guides the way to why, where and when

• Last release 2007 (nowadays a package with Dyninst)
Paradyn – overview (2)

• Platforms / Environments: Solaris, Linux, AIX, Windows and MPI

• Since it instruments binaries (executable files) it is fully programming language independent
  – Interpreted languages will not work properly since it can only instrument the interpreter
Instrumentation

- How does instrumentation influence normal program execution?
  - Execution of instrumentation code (e.g. data capturing calls) takes more than a certain amount of time
  - Instrumenting code changes compiler behavior:
    - Can block compiler optimizations
- Instrumentation detail vs. perturbation
- Changed caching behavior
- Usually 1-5% perturbation acceptable
Hardware and software performance metrics (1)

• Software
  – Calls made by the program during execution to gather data
    • Process time
    • Wall time

• Hardware
  – Needed if some machine behavior is hardware controlled (e.g. caching strategy) and abstracted (can not seen by the sw layer)
  – Realized through hardware counters in modern microprocessors
    • L1 and L2 cache misses
    • Count cycles
    • Floating point instructions per time unit
    • Branch mispredictions
Hardware and software performance metrics (2) - examples

- **Timers**
  - Process time
  - Time spent for executing a particular function
  - Wall time
  - Wait time for message receives

- **Counters**
  - Count calls of a particular function
  - Hardware counters
  - Count the use of synchronization objects (e.g. barriers)
SvPablo: Performance Instrumentation

- Allows *interactive* and *automatic* instrumentation
  - Interactive (via instrumenting parsers)
    - Inserts calls in source code following certain rules / patterns:
      - outer loops
      - procedure calls
    - Lets the user modify these calls (add / delete calls)
    - Possible drawbacks: excessive perturbation, inhibition of compiler optimizations
  - Automatic (only for HPF)
    - Inserts probes in compiler-synthesized code (one level deeper)
      - FORALL(I=1:N) A(I)= B(I)
    - Reduces the probability of excessive instrumentation perturbation (by simply forbidding it)
    - Possible drawbacks: no explicit user control, still inhibition of compiler optimizations
SVPablo: Measuring

• Software and Hardware performance data
  – Hardware counters can be added in an ASCII file
  – Additionally computation of derived metrics:
    • Percentages of computation time
SvPablo: Interpreting

Menu bar

List of source files

List of routines in a source file

Source file display

List of performance contexts

List of routines in performance file
SvPablo: Interpreting data

Procedure level performance data

Colored boxes represent the count and exclusive duration
SvPablo: Interpreting data

- Via visualization
- Paper example
- The two right most dark bars show communication metrics
- Problem is the cshift operation
- Can be prefetched because s and kmh are loop independent
- flx is not
- Solution: create 4 independent loops
- 197 sec -> 14.5 sec
- Poorly presented example
Paradyn: Performance Instrumentation

• Instruments code during running time by modifying the running program
  – Can add and delete instrumentation at runtime
  – This feature is used to apply a finer grained instrumentation only to parts of interest
    • Advantage: less perturbation
  – Can attach to an already running program
  – Focuses on long time runners
    • Short time runners can be instrumented by running them multiple times
Paradyn: measuring
Paradyn: measuring $W^3$ search model (1)

• Basic idea: some intelligent logic has to decide
  – At which points to instrument?
  – What to instrument?
  – Remember goal: keep instrumentation perturbation small

• Function of the performance consultant

• The $W^3$ space:
  – 1st Dim: Why
    • potential performance problems
  – 2nd Dim: Where
    • program components
  – 3rd Dim: When
    • a series of distinct phases
Paradyn: measuring $W^3$ search model (2)

• Why:
  – A list of hypothesis
  – A set of tests to evaluate the validity of each of them
Paradyn: measuring \( W^3 \) search model (3)

- Where:
  - A list of system components
Paradyn: measuring $W^3$ search model (3)

- When:
  - Time
  - Select a “why” and a “where”
Paradyn: measuring search algorithm
Paradyn: measuring critics

• How hard is the job of creating the right set of why-hypothesis and especially their thresholds
• It is not clear which of the possible true why-hypothesis to investigate further first.
• Since all instrumentation dynamically changes what happens if we are measuring the wrong metrics at the wrong place the wrong time?
• How do we discover several performance bottlenecks?
• The algorithm searches only in the why-where space (not as claimed in the $W^3$ space)
What SvPablo can – and Paradyn cannot

• Running the same program in different contexts and visualize them
  – 2,4,8,16 processors
  – Different machines
  – Scalability analysis

• Works well with small applications
SvPablo: Scaleability

Routines with the worst scalability listed at the top

The line graph shows the efficiency drops for med_shutdown_IO
What Paradyn can – and SvPablo cannot

• Hooking up to a running system
  – Running parallel DB server
  – Can watch an application for some time
• Instrument long time runners
• Data interpretation included
Open discussion (1)

- Paradyn tries to interpret performance results to give the user a complete answer
- SvPablo provides a transparent measurement plus visualization

- What works better in practice?
Conclusions / Discussion (2)

• SvPablo lets the user instrument with the risk of measurement perturbation
• Paradyn instruments code on the fly to minimize measurement perturbation

• Is it worth the effort being made to implement this?
Questions?
Thank you.