HPCToolkit: Multi-platform Tools for Profile-based Performance Analysis

John Mellor-Crummey

Robert Fowler    Nathan Tallent    Gabriel Marin

Department of Computer Science
Rice University

http://hipersoft.cs.rice.edu/hpctoolkit/
Performance Analysis and Tuning

• Increasingly necessary
  — gap between typical and peak performance is growing

• Increasingly hard
  — complex architectures are harder to program effectively
    – complex processors
      VLIW
deepl pipelined, out of order, superscalar
    – complex memory hierarchy
      non-blocking, multi-level caches
      TLB
  — modern scientific applications pose challenges for tools
    – multi-lingual programs
    – many source files
    – complex build process
    – external libraries in binary-only form
HPCToolkit Goals

- Support large, multi-lingual applications
  - a mix of Fortran, C, C++
  - external libraries
  - thousands of procedures
  - hundreds of thousands of lines
  - we must avoid
    - manual instrumentation
    - significantly altering the build process
    - frequent recompilation

- Multi-platform
- Scalable data collection
- Analyze both serial and parallel codes
- Effective presentation of analysis results
  - intuitive enough for physicists and engineers to use
  - detailed enough to meet the needs of compiler writers
HPCToolkit System Overview

- **Application Source**
- **Compilation**
- **Linking**
- **Binary Object Code**
- **Source Correlation**
- **Binary Analysis**
- **Program Structure**
- **Profile Execution**
- **Profile Interpretation**
- **Performance Profile**
- **Hyperlinked Database**
- **HPCViewer**
HPCToolkit System Overview

- launch unmodified, optimized application binaries
- collect statistical profiles of events of interest

http://hipersoft.cs.rice.edu/hpctoolkit/
HPCToolkit System Overview

- application source
  - compilation
  - linking
  - binary object code
  - binary analysis
  - program structure
  - interpret profile
  - hpcviewer
  - hyperlinked database
  - source correlation

- decode instructions and combine with profile data

http://hipersoft.cs.rice.edu/hpctoolkit/
HPCToolkit System Overview

- application source
- compilation
- linking
- binary object code
- binary analysis
- program structure
- interpret profile
- profile execution
- performance profile
- hyperlinked database
- source correlation
- hpcviewer

extract loop nesting information from executables
HPCToolkit System Overview

—synthesize new metrics by combining metrics
—relate metrics, structure, and program source
HPCToolkit System Overview

- support top-down analysis with interactive viewer
- analyze results anytime, anywhere
Data Collection

Support analysis of unmodified, optimized binaries

• Inserting code to start, stop and read counters has many drawbacks, so don’t do it!
  —nested measurements skew results

• Use hardware performance monitoring to collect statistical profiles of events of interest

• Different platforms have different capabilities
  —event-based counters: MIPS, IA64, Pentium
  —ProfileMe instruction tracing: Alpha

• Different capabilities require different approaches
Data Collection Tools

Goal: limit development to essentials only

- **MIPS-IRIX:**
  - ssrun + prof → ptran

- **Alpha-Tru64:**
  - uprofile + prof → ptran
  - DCPI/ProfileMe → xprof

- **IA64-Linux and IA32-Linux**
  - papirun/papipprof
papirun/papiprof

• PAPI: Performance API
  — interface to hardware performance monitors
  — supports many platforms

• papirun: open source equivalent of SGI’s ‘ssrun’
  — sample-based profiling of an execution
    – preload monitoring library before launching application
    – inspect load map to set up sampling for all load modules
    – record PC samples for each module along with load map
  — Linux IA64 and IA32

• papiprof: ‘prof’-like tool
  — based on Curtis Janssen’s vprof
  — uses GNU binutils to perform PC \(\rightarrow\) source mapping
  — output styles
    – XML for use with hpcview
    – plain text
DCPI and ProfileMe

• Alpha ProfileMe
  — EV67+ records info about an instruction as it executes
    – mispredicted branches, memory access replay traps
    – more accurate attribution of events

• DCPI: (Digital) Continuous Profiling Infrastructure
  — sample processor counters and instructions continuously during execution of all code
    – all programs
    – shared libraries
    – operating system
  — support both on-line and off-line data analysis
    – to date, we use only off-line analysis
HPCToolkit System Overview

1. Application source
2. Compilation
3. Linking
4. Binary object code
5. Binary analysis
6. Program structure
7. Interpret profile
8. Profile execution
9. Performance profile
10. Hyperlinked database
11. Source correlation
12. Hpcviewer
Metric Synthesis with xprof (Alpha)

Interpret DCPI samples into useful metrics

• Transform low-level data to higher-level metrics
  — DCPI ProfileMe information associated with PC values
  — project ProfileMe data into useful equivalence classes
  — decode instruction type info in application binary at each PC
    - FLOP
    - memory operation
    - integer operation
  — fuse the two kinds of information
    - Retired instructions + instruction type =
      retired FLOPs
      retired integer operations
      retired memory operations

• Map back to source code like papiprof

http://hipersoft.cs.rice.edu/hpctoolkit/
HPCToolkit System Overview

1. Application source
2. Compilation
3. Linking
4. Binary object code
5. Binary analysis
6. Program structure
7. Interpret profile
8. Profile execution
9. Performance profile
10. Hyperlinked database
11. Source correlation
12. Hpcviewer
Program Structure Recovery with bloop

- Parse instructions in an executable using GNU binutils
- Analyze branches to identify basic blocks
- Construct control flow graph using branch target analysis —be careful with machine conventions and delay slots!
- Use interval analysis to identify natural loop nests
- Map machine instructions to source lines with symbol table —dependent on accurate debugging information!
- Normalize output to recover source-level view

Platforms: Alpha+Tru64, MIPS+IRIX, Linux+IA64, Linux+IA32, Solaris+SPARC
Sample Flowgraph from an Executable

Loop nesting structure
— blue: outermost level
— red: loop level 1
— green loop level 2

Observation: optimization complicates program structure!
Normalizing Program Structure

**Constraint:** each source line must appear at most once

**Coalesce duplicate lines**

1. **if duplicate lines appear in different loops**
   - find least common ancestor in scope tree; merge corresponding loops along the paths to each of the duplicates
   - **purpose:** re-rolls loops that have been split

2. **if duplicate lines appear at multiple levels in a loop nest**
   - discard all but the innermost instance
   - **purpose:** handles loop-invariant code motion

apply (1) and (2) repeatedly until a fixed point is reached
Recovered Program Structure

```xml
<LM n="/apps/smg98/test/smg98">
...<F n="/apps/smg98/struct_linear_solvers/smg_relax.c">
<P n="hypre_SMGRelaxFreeARem">
<L b="146" e="146">
<S b="146" e="146"/>
</L></P>
</P>
<P n="hypre_SMGRelax">
<L b="297" e="328">
<S b="297" e="297"/>
<L b="301" e="328">
<S b="301" e="301"/>
<L b="318" e="325">
<S b="318" e="325"/>
</L>
<S b="328" e="328"/>
</L>
<S b="302" e="302"/>
</L>
</P>
...</F></PGM>
```
HPCToolkit System Overview

- Application source
  - Compilation
  - Linking
  - Source correlation
  - Hyperlinked database
    - Hyperlinked database
      - Hpcviewer

- Binary object code
  - Binary analysis
    - Program structure
      - Interpret profile
        - Performance profile

http://hipersoft.cs.rice.edu/hpctoolkit/
Data Correlation

• Problem
  — any one performance measure provides a myopic view
    – some measure potential *causes* (e.g. cache misses)
    – some measure *effects* (e.g. cycles)
    – cache misses not always a problem
  — event counter attribution is inaccurate for out-of-order processors

• Approaches
  — multiple metrics for each program line
  — computed metrics, e.g. cycles - FLOPS
    – eliminate mental arithmetic
    – serve as a key for sorting
  — hierarchical structure
    – line level attribution errors give good loop-level information
HPCToolkit System Overview

application source -> compilation linking -> binary object code

source correlation -> binary analysis

hyperlinked database -> interpret profile

program structure

profile execution -> performance profile

hpcviewer

http://hipersoft.cs.rice.edu/hpctoolkit/
HPCViewer Screenshot

Annotated Source View

Navigation

Metrics
Flattening for Top Down Analysis

• Problem
  — strict hierarchical view of a program is too rigid
  — want to compare program components at the same level as peers

• Solution
  — enable a scope’s descendants to be flattened to compare their children as peers

http://hipersoft.cs.rice.edu/hpctoolkit/
Some Uses for HPCToolkit

• Identifying unproductive work
  —where is the program spending its time not performing FLOPS

• Memory hierarchy issues
  —bandwidth utilization: misses x line size/cycles
  —exposed latency: ideal vs. measured

• Cross architecture or compiler comparisons
  —what program features cause performance differences?

• Gap between peak and observed performance
  —loop balance vs. machine balance?

• Evaluating load balance in a parallelized code
  —how do profiles for different processes compare
Assessment of HPCToolkit Functionality

• Top down analysis focuses attention where it belongs
  —sorted views put the important things first
• Integrated browsing interface facilitates exploration
  —rich network of connections makes navigation simple
• Hierarchical, loop-level reporting facilitates analysis
  —more sensible view when statement-level data is imprecise
• Binary analysis handles multi-lingual applications and libraries
  —succeeds where language and compiler based tools can’t
• Sample-based profiling, aggregation and derived metrics
  —reduce manual effort in analysis and tuning cycle
• Multiple metrics provide a better picture of performance
• Multi-platform data collection
• Platform independent analysis tool
What’s Next?

Research

— collect and present dynamic content
  – what path gets us to expensive computations?
  – accurate call-graph profiling of unmodified executables
  – analysis and presentation of dynamic content
— communication in parallel programs
— statistical clustering for analyzing large-scale parallelism
— performance diagnosis: why rather than what

Development

— harden toolchain
— new platforms: Opteron and PowerPC
— data collection with oprofile on Linux