Performance Analysis of MPI+OpenMP Programs with HPCToolkit

John Mellor-Crummey
Department of Computer Science
Rice University

http://hpctoolkit.org
Acknowledgments

• **Project team**
  — **Research Staff**
    – Laksono Adhianto, Mike Fagan, Mark Krentel
  — **Students**
    – Milind Chabbi, Karthik Murthy
  — **Recent Alumni**
    – Xu Liu (William and Mary, 2014)
    – Nathan Tallent (PNNL, 2010)

• **Current funding**
  — DOE Office of Science ASCR X-Stack “PIPER” Award
  — Intel
  — BP (pledge)
Challenges for Computational Scientists

• Rapidly evolving platforms and applications
  — architecture
    – rapidly changing multicore microprocessor designs
    – increasing architectural diversity
      multicore, manycore, accelerators
    – increasing scale of parallel systems
  — applications
    – transition from MPI everywhere to threaded implementations
    – enhance vector parallelism
    – augment computational capabilities

• Computational scientists needs
  — adapt to changes in emerging architectures
  — improve scalability within and across nodes
  — assess weaknesses in algorithms and their implementations

Performance tools can play an important role as a guide
Performance Analysis Challenges

• Complex node architectures are hard to use efficiently
  — multi-level parallelism: multiple cores, ILP, SIMD, accelerators
  — multi-level memory hierarchy
  — result: gap between typical and peak performance is huge

• Complex applications present challenges
  — measurement and analysis
  — understanding behaviors and tuning performance

• Multifaceted performance concerns
  — computation
  — data movement
  — communication
  — I/O
What Users Want

- Multi-platform, programming model independent tools
- Accurate measurement of complex parallel codes
  - large, multi-lingual programs
  - (heterogeneous) parallelism within and across nodes
  - optimized code: loop optimization, templates, inlining
  - binary-only libraries, sometimes partially stripped
  - complex execution environments
    - dynamic binaries on clusters
    - static binaries on supercomputers
    - batch jobs
- Effective performance analysis
  - insightful analysis that pinpoints and explains problems
    - correlate measurements with code for actionable results
    - support analysis at the desired level
      - intuitive enough for application scientists and engineers
      - detailed enough for library developers and compiler writers
- Scalable to large jobs
Outline

• Overview of Rice’s HPCToolkit

• Pinpointing scalability bottlenecks
  — scalability bottlenecks on large-scale parallel systems
  — scaling on multicore processors

• Understanding temporal behavior

• Assessing variability across ranks and threads

• Understanding threading performance
  — blame shifting

• A tuning strategy

• Putting it all together
  — analyze an execution of a DRTM code (48 MPI ranks x 6 OpenMP)

• Ongoing work and future plans

• For your reference: getting and using HPCToolkit
Rice University’s HPCToolkit

- Employs binary-level measurement and analysis
  - observe fully optimized, dynamically linked executions
  - support multi-lingual codes with external binary-only libraries

- Uses sampling-based measurement (avoid instrumentation)
  - controllable overhead
  - minimize systematic error and avoid blind spots
  - enable data collection for large-scale parallelism

- Collects and correlates multiple derived performance metrics
  - diagnosis typically requires more than one species of metric

- Associates metrics with both static and dynamic context
  - loop nests, procedures, inlined code, calling context

- Supports top-down performance analysis
  - identify costs of interest and drill down to causes
    - up and down call chains
    - over time
HPCToolkit Workflow

source code → optimized binary

compile & link

profile execution
[hpcrun]

call path profile

binary analysis
[hpcstruct]

program structure

interpret profile correlate w/ source
[hpcprof/hpcprof-mpi]

presentation
[hpcviewer/ hpctraceviewer]

database

HPCToolkit Workflow
HPCToolkit Workflow

- For dynamically-linked executables, e.g., Linux
  — compile and link as you usually do: nothing special needed*

  * Note: OpenMP currently requires a special enhanced runtime for tools to be added at link time or program launch
Measure execution unobtrusively
- launch optimized application binaries
  - dynamically-linked: launch with `hpcrun`, arguments control monitoring
  - collect statistical call path profiles of events of interest
Call Path Profiling

Measure and attribute costs in context

- sample timer or hardware counter overflows
- gather calling context using stack unwinding

Call path sample

- return address
- return address
- return address
- instruction pointer

Overhead proportional to sampling frequency...
...not call frequency
HPCToolkit Workflow

- Analyze binary with `hpcstruct`: recover program structure
  - analyze machine code, line map, debugging information
  - extract loop nesting & identify inlined procedures
  - map transformed loops and procedures to source

presentation
  [hpcviewer/hpctraceviewer]

interpret profile correlate w/ source
  [hpcprof/hpcprof-mpi]
database

compile & link

source code

optimized binary

profile execution
  [hpcrun]
call path profile

program structure

binary analysis
  [hpcstruct]
HPCToolkit Workflow

- Combine multiple profiles
  — multiple threads; multiple processes; multiple executions
- Correlate metrics to static & dynamic program structure
HPCToolkit Workflow

- **Presentation**
  - explore performance data from multiple perspectives
    - rank order by metrics to focus on what’s important
    - compute derived metrics to help gain insight
      - e.g. scalability losses, waste, CPI, bandwidth
  - graph thread-level metrics for contexts
  - explore evolution of behavior over time
Code-centric Analysis with hpcviewer

- inlined procedures
- loops
- function calls in full context

Source pane

View control

Metric display

Navigation pane

Metric pane
The Problem of Scaling

Note: higher is better
Goal: Automatic Scaling Analysis

- Pinpoint scalability bottlenecks
- Guide user to problems
- Quantify the magnitude of each problem
- Diagnose the nature of the problem
Challenges for Pinpointing Scalability Bottlenecks

• Parallel applications
  — modern software uses layers of libraries
  — performance is often context dependent

• Monitoring
  — bottleneck nature: computation, data movement, synchronization?
  — 2 pragmatic constraints
    – acceptable data volume
    – low perturbation for use in production runs
Performance Analysis with Expectations

• You have performance expectations for your parallel code
  — strong scaling: linear speedup
  — weak scaling: constant execution time

• Put your expectations to work
  — measure performance under different conditions
    – e.g. different levels of parallelism or different inputs
  — express your expectations as an equation
  — compute the deviation from expectations for each calling context
    – for both inclusive and exclusive costs
  — correlate the metrics with the source code
  — explore the annotated call tree interactively
Pinpointing and Quantifying Scalability Bottlenecks

\[
\frac{1}{Q} \times 600K - \frac{1}{P} \times 400K = \frac{1}{P} \times 200K
\]

coefficients for analysis of weak scaling
Parallel, adaptive-mesh refinement (AMR) code

- Designed for compressible reactive flows
- Can solve a broad range of (astro)physical problems
- Portable: runs on many massively-parallel systems
- Scales and performs well
- Fully modular and extensible: components can be combined to create many different applications

Scalability Analysis Demo

**Code:**
- University of Chicago FLASH

**Simulation:**
- white dwarf detonation

**Platform:**
- Blue Gene/P

**Experiment:**
- 8192 vs. 256 processors

**Scaling type:**
- weak

Figures courtesy of FLASH Team, University of Chicago
Scalability Analysis of Flash (Demo)
Scalability Analysis

• Difference call path profile from two executions
  — different number of nodes
  — different number of threads

• Pinpoint and quantify scalability bottlenecks within and across nodes

significant scaling losses caused by passing data around a ring of processors
Improved Flash Scaling of AMR Setup

Graph courtesy of Anshu Dubey, U Chicago
Profiling compresses out the temporal dimension
—temporal patterns, e.g. serialization, are invisible in profiles

What can we do? Trace call path samples
—sketch:
  – N times per second, take a call path sample of each thread
  – organize the samples for each thread along a time line
  – view how the execution evolves left to right
  – what do we view?
    assign each procedure a color; view a depth slice of an execution
Trace View of FLASH3@256PE (Demo)

Time-centric analysis: load imbalance among threads appears as different lengths of colored bands along the x axis
OpenMP: A Challenge for Tools

- Large gap between between threaded programming models and their implementations

User-level calling context for code in OpenMP parallel regions and tasks executed by worker threads is not readily available

- Runtime support is necessary for tools to bridge the gap
Challenges for OpenMP Node Programs

• Typically, tools present an implementation-level view of OpenMP threads
  — asymmetric threads
    – master thread
    – worker thread
  — run-time frames are interspersed with user code

• Hard to understand relationship to program structure

• Hard to understand causes of idleness
  — serial sections
  — load imbalance in parallel regions
  — waiting for critical sections or locks
OMPT: An OpenMP Tools API

• Goal: a standardized tool interface for OpenMP
  — prerequisite for portable tools
  — missing piece of the OpenMP language standard

• Design objectives
  — enable tools to measure and attribute costs to application source and runtime system
    • support low-overhead tools based on asynchronous sampling
    • attribute to user-level calling contexts
    • associate a thread’s activity at any point with a descriptive state
  — minimize overhead if OMPT interface is not in use
    • features that may increase overhead are optional
  — define interface for trace-based performance tools
  — don’t impose an unreasonable development burden
    • runtime implementers
    • tool developers
OpenMP Tools API Status

• April 2014: OpenMP TR2
  — OMPT: An OpenMP Tools Application Programming Interface for Performance Analysis
    – Alexandre Eichenberger (IBM), John Mellor-Crummey (Rice), Martin Schulz (LLNL) et al
  — major step toward having a tools API added to OpenMP standard

• OMPT implementations
  — IBM, Intel (prototype), LLVM (coming)

• Next steps
  — transition OMPT prototype into Intel for use with production OpenMP runtime
  — propose OMPT additions to the language standard
Analyzing MPI+OpenMP with OMPT (Demo)

AMG2006: 4 MPI ranks x (8 OpenMP threads + 3 helper threads)
<table>
<thead>
<tr>
<th>Problem</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Undirected Blame Shifting</strong>&lt;sup&gt;1,3&lt;/sup&gt;</td>
<td>A thread is idle waiting for work</td>
</tr>
<tr>
<td><strong>Directed Blame Shifting</strong>&lt;sup&gt;2,3&lt;/sup&gt;</td>
<td>A thread is idle waiting for a mutex</td>
</tr>
</tbody>
</table>

1. Tallent & Mellor-Crummey: PPoPP 2009
2. Tallent, Mellor-Crummey, Porterfield: PPoPP 2010
3. Liu, Mellor-Crummey, Fagan: ICS 2013
Blame-shifting Metrics for OpenMP

- **OMP_IDLE**
  - attribute idleness to insufficiently-parallel code being executed by other threads

- **OMP_MUTEX**
  - attribute waiting for locks to code holding the lock
    - attribute to the lock release as a proxy

- Measuring these metrics requires sampling using using a time-based sample source
  - REALTIME, CPUTIME, PAPI_TOT_CYC
Blame Shifting with AMG2006 (Demo)

AMG2006: 4 MPI ranks x (8 OpenMP threads + 3 helper threads)
Assessing Variability (Demo)

AMG2006: 4 MPI ranks x (8 OpenMP threads + 3 helper threads)
A Recipe for Tuning MPI + OpenMP

• In priority order
  – get the large-scale MPI parallelization right
    • if processes are blocked, performance will be lost
  – get the OpenMP threading right
    • if threads are blocked, performance will be lost
  – get the node performance details right
    • assess memory hierarchy performance (TLB, cache)
    • assess pipeline performance (graduated instructions, …)
Putting it all Together (DRTM)

DRTM code: 48 MPI ranks x (6 OpenMP threads/rank + 3 helper threads)
Other HPCToolkit Capabilities

- **Performance analysis of GPU-accelerated code**

- **Data-centric performance analysis**
  - Xu Liu and John Mellor-Crummey, "A Tool to Analyze the Performance of Multithreaded Programs on NUMA Architectures" PPoPP’14, Feb, 2014, Orlando, Florida, USA.
  - Xu Liu and John Mellor-Crummey, "A Data-centric Profiler for Parallel Programs" SC13, Nov. 2013, Denver, Colorado, USA.
Ongoing Work and Future Plans

• Ongoing work
  — refining support for OMPT in HPCToolkit and OpenMP runtime
  — refining measurement, analysis, and attribution
    — optimized code
    — general multithreaded models, e.g., TBB, CilkPlus
  — improving scalability of hpctraceviewer and server

• Plans
  — enhanced performance analysis of GPU-accelerated code
    – sampling-based measurement on emerging NVIDIA GPUs
  — resource-centric performance analysis
    – e.g., bandwidth: I/O, communication, memory
  — refined data-centric analysis: GUI to attribute costs to data
  — measurement and analysis for exascale
  — automated analysis to deliver insights
For Your Reference

Getting and Using HPCToolkit
Getting HPCToolkit

• Open source software. See hpctoolkit.org for pointers
• See hpctoolkit.org for instructions to download and build
• Three different pieces of HPCToolkit
  — hpctoolkit-externals
    – source code available in an svn repository on google code
  — hpctoolkit
    – source code available in an svn repository on google code
    – OMPT support is still in a branch
      svn co http://hpctoolkit.googlecode.com/svn/branches/hpctoolkit-ompt
  — hpcviewer and hpctraceviewer user interfaces
    – binary packages for your laptop, workstation, or cluster
      http://hpctoolkit.org/download/hpcviewer
        hpcviewer and hpctraceviewer
        linux, mac, and windows binaries
    – source code available for a Java Eclipse RCP project
• Useful external library: PAPI for measuring hardware counters
  — http://icl.cs.utk.edu/papi
Detailed HPCToolkit Documentation

http://hpctoolkit.org/documentation.html

• User manual:
  
  
  — Quick start guide
  
  — Using HPCToolkit with statically linked programs
  
  — The hpcviewer and hpctraceviewer user interfaces
  
  — Effective strategies for analyzing program performance with HPCToolkit
  
  — HPCToolkit and MPI
  
  — HPCToolkit Troubleshooting
    
    — why don’t I have any source code in the viewer?
    — hpcviewer isn’t working well over the network ... what can I do?

• Installation guide
Getting OMPT-enhanced Intel OpenMP

- Currently a prototype open source project
  - https://code.google.com/p/ompt-intel-openmp
- Soon will be provided to Intel for integration in their runtime
- Getting the prototype
  - clone the git repository with the code
    - git clone https://code.google.com/p/ompt-intel-openmp
    - cd ompt-intel-openmp
    - git checkout ompt-support-14x
    - cd itt/libompss
    - make
    - the resulting runtime, with OMPT support, will be in the exports directory
Using HPCToolkit

• Adjust your compiler flags (if you want full attribution to src)
  — add -g flag after any optimization flags

• See what sampling triggers are available on your platform
  — hpcrun -L
  — If your system’s login nodes are different, you need to run this command on your compute nodes
Collecting Performance Data

• Collecting traces
  — use a time-based sample source when collecting a trace
    – CPUPTIME, REALTIME, PAPI_TOT_CYC
  — use the -t option to hpcrun

• Measuring threads
  — use REALTIME to profile threads
    – otherwise you miss when they sleep
    – need to use HPCRUN_IGNORE_THREAD=1
      need to ignore OpenMP (+ MPI) helper threads

• Measuring an MPI job using hpcrun
  — change
    mpiexec -np 4 your_program arguments
  — to
    mpiexec -np 4 \n      hpcrun -e REALTIME@1000 -e OMP_IDLE -t \n      your_program arguments
Digesting your Performance Data

• Use hpcstruct to reconstruct program structure
  — e.g. hpcstruct your_app
    – creates your_app.hpcstruct

• Correlate measurements to source code
  — hpcprof
    – use on a workstation to analyze data from modest runs
  — hpcprof-mpi
    – use on a cluster’s compute nodes to analyze data in parallel from lots of nodes/threads
Analysis and Visualization

• Use hpcviewer to open resulting database
  — warning: first time you graph any data, it will pause to combine info from all threads into one file

• Use hpctraceviewer to explore traces
  — warning: first time you open a trace database, the viewer will pause to combine info from all threads into one file

• Try our our user interfaces before collecting your own data
  — example performance data at http://hpctoolkit.org/examples.html
Monitoring Large Executions

• Collecting performance data on every node is typically not necessary

• Can improve scalability of data collection by recording data for only a fraction of processes
  — set environment variable HPCRUN_PROCESS_FRACTION
  — e.g. collect data for 10% of your processes
    – set environment variable HPCRUN_PROCESS_FRACTION=0.10
Tuning Recipe for MPI + OpenMP - I

Get the large-scale MPI parallelization right first

• Use an appropriate domain decomposition
  – balance load
  – consider communication frequency and volume
    • avoid excessive fine-grain messages
  – avoid serialization
  – make sure that parallelism is available on the node as well for use with OpenMP

• Use asynchronous communication primitives where possible
  – make computation asynchrony tolerant
    • overlap communication with computation

• Tools
  – use hpcviewer to look for performance and scaling bottlenecks
    • issues apparent within a single execution
    • comparative analysis of multiple executions (strong or weak scaling)
  – use hpctraceviewer to understand MPI parallelization
Get the OpenMP threading right

- Employ OpenMP where appropriate
  - avoid fine-grain parallel regions and loop nests
  - barriers at the end of loops and regions can be costly
  - consider how load will be balanced between threads

- Consider OpenMP tasking for functional parallelism

- Tools
  - use hpcviewer and hpctraceviewer to examine threading performance
    - the summary view can help you assess idleness
Get the node performance right

- Use hpcrun to profile your code using hardware performance counters
  - measure resource stalls and compare them with instruction and cycle counts
  - measure the memory hierarchy performance
    - caches and TLB
  - assess vector vs. scalar code
    - vectors are an opportunity to accelerate your code
  - see the HPCToolkit manual for how to compute useful “waste” metrics

- Tools
  - use hpcviewer to assess node performance at the call path, function, and loop levels