Scalable Automated Online Performance Analysis of Applications using Performance Properties

Karl Fürlinger, Michael Gerndt
{Karl.Fuerlinger, Michael.Gerndt}@in.tum.de

Lehrstuhl für Rechnertechnik und Rechnerorganisation,
Institut für Informatik
Technische Universität München (TUM)
Outline

- Motivation
- Automating Performance Analysis with **Performance Properties**

  **Periscope**: distributed, agent-based tool for automatically locating bottlenecks in MPI and OpenMP applications using performance properties

- Usage Examples and Results
  - APART Test Suite
  - NAS Parallel Benchmarks

- Summary
Motivation (1)

- Supercomputers are getting bigger and bigger
  - Several thousand processors are commonplace in HPC systems
  - BlueGene/L: 128,000 processors

- Multicore CPUs are accelerating this trend
  - Performance per core stays approximately constant => overall performance increases through higher levels of parallelism

- Performance analysis necessary to ensure that resources are not wasted

- Large quantities of performance data
  - Tracing can give gigabytes of performance data
  - Too much data for manual analysis

- Not every programmer can be an expert
  - Or wants to be…
  - For every machine…

- Automation can solve some of those problems
Motivation (2)

- **Idea:** Automating the process of performance analysis
  - Automatically detecting performance problems (and potentially listing reasons and remedies)
  - Expert knowledge is embedded in the performance tool

- **Further advantage:** Analysis becomes independent of time and space
  - Analysis can happen at any time
  - Analysis can happen in a distributed and parallel way
Approaches for Automated Performance Analysis

- Expert:
  - Automated search for Performance problems in program traces
  - Post-mortem

- Paradyn/Performance Consultant:
  - Dynamic instrumentation
  - W3 Search model (why/where/when)
  - Online

- Aksum:
  - JavaPSL, profile-based performance data stored in a database
  - Automated search for performance problems
  - Post-mortem

- Our Approach with Periscope:
  - Based on ASL performance properties
  - Online
One approach to make expert knowledge available to automated performance tools.

**ASL: APART Specification Language**
- Approach to formally specify situations of inefficient program execution
- Two constituents:
  1. **Data Model**: Describes the data that becomes available from a monitored execution of the target application
  2. **Performance Properties**: Description of performance problems, using the data model

**Performance Properties: Three main parts:**
- **Condition**: Method to determine if a certain situation of inefficient program execution exists.
- **Severity**: Quantifies how severe the problem is, how much performance is lost
- **Confidence**: quantifies the certainty in the detection of the performance problem.
Example: parallel region with 4 threads, load is imbalanced

**exitBarT[i]** time that thread i spends in the “exit barrier”, quantifies load imbalance

Property ImabalanceInParallelRegion
{
  let
    imbal=exitBarT[0]+...+exitBarT[N-1];

  Condition: (region.type)==PARALLEL && (imbal>0.0);
  Severity: imbal/(total runtime * number of threads);
  Confidence: 1.0;
}
Several timers and counters, similar to `exitBarT`, depending on the particular OpenMP construct.

```c
ParPerf {
    Region *reg       // The region for which the summary is collected
    Experiment *exp  // The experiment where this data belongs to
    int threadC       // Number of threads that executed the region
    double execT[]    // Total execution time per thread
    double execC[]    // Total execution count per thread
    double exitBarT[]  // Time spent in the implicit exit barrier
    double singleBodyT[] // Time spent inside a single construct
    int singleBodyC[]  // Execution count in a single construct
    double enterT[]    // Time spent waiting to enter a construct
    int enterC[]       // Number of times a threads enters a construct
    double exitT[]     // Time spent to exit a construct
    int exitC[]        // Number of times a thread exits a construct
    double sectionT[]  // Time spent inside a section construct
    int sectionC[]     // Number of times a section construct is entered
    double startupT[]  // Time required to create a parallel region
    double shutdownT[] // Time required to destroy a parallel region
}
```
Data Model for OpenMP (2)

- ParPerf structure based on what we can practically measure
  - Instrumentation with Opari

- Not all members defined for all region types.
  - `singleBodyT`, `singleBodyC` only for `single` regions
  - `exitBarT` measures time in the implicit exitbarrier barrier added by Opari in worksharing and parallel regions
  - `enterT`, `exitT`, `enterC`, `exitC` time for entering and exiting `locks` or `critical` sections
  - `sectionT`, `sectionC` time inside `section` constructs
  - `startupT`, `shutdownT`: startup and teardown overhead for parallel regions
Further Properties (OpenMP)

- **ImbalanceInParallelRegion:**
  - A parallel region contains an imbalanced distribution of work, some threads finish their work earlier than others.
  - Similar properties: ImbalanceInParallelLoop, ImbalanceInParallelSections, ImbalanceInParallelWorkshare

- **CriticalSectionContention**
  - Several Threads try to enter a critical section at the same time, only one thread can be inside the critical section at any time, the other threads experience waiting time.
  - Similar property: LockContention

- **UnparallelizedInSingleRegion**
  - A program region is only executed by one thread, other threads have to wait.
  - Similar property: UnparallelizedInMasterRegion

- **ImbalanceDueToUnevenSectionDistribution, ImbalanceDueToNotEnoughSections**

- ...
**Further Properties (MPI)**

- **Data model**
  - Currently only summaric data: bytes transmitted, number of calls, total time

  ```
  MpiPerf {
    Region *reg; // The MPI call for which the summary is collected
    double execT; // The summed execution time for the MPI call
    int execC; // The summed execution count for the MPI call
    int msgLen; // The summed size of sent or received messages (in bytes)
  }
  ```

- **Example:**

  ```
  PROPERTY WaitAtMpiBarrier(MpiPerf pd, Experiment exp) {
    LET
      double waittime = pd.execT;
      
      IN
      condition : (pd.reg->type==MPI_BARRIER) &&
      (waittime > 0.0);
      confidence : 1.0;
      severity : waittime / RB(exp);
  }
  ```

- **Other properties:** CollectiveCommOvhd, PointToPointCommOvhd, LargeMessageSize, FrequentCommunication
Periscope (1)

- System that automatically searches for performance properties

- Target application: written in C/C++/Fortran, parallelized using OpenMP, MPI or combinations

- Not a monolithic tool, but composed of several distributed and cooperating components (called agents)
Components of Periscope

- **Node-Level Agent:**
  - Analyzes performance data, searches for performance properties

- **High-Level Agent:**
  - Tree-like hierarchy of agents
  - Combination of results of the Node-level agents, distribution of commands

- **Master Agent:**
  - Root of the tree, top of the hierarchy
  - Establishes the connection to the frontend component

- **Frontend:**
  - User-interface, takes commands, displays performance analysis results

- **Registry Service:**
  - Central part used during tool startup by the other components to discover each other and to establish communication links
Usage Scenario

Property 'ImbalanceAtMPIBarrier' holds for 'region line-number' on 'host1, host2, …' severity 0.44201, confidence 1.0
Monitoring is realized by “common memory” segments
- Contains configuration tables and ring-buffers holding run-time events
- Single events, (e.g., “enter region”) are stored to thread-local buffers
- Implementation is either based on actual shared memory or realized via RDMA (remote direct memory access)

The agents communicate using the ACE (adaptive communications environment) framework

Clustering of properties:

- WaitAtBarrier "opt33" 0.34 "ssor.f 186"
- WaitAtBarrier "opt33" 0.33 "ssor.f 207"
- WaitAtBarrier "opt33,opt34" 0.32 "ssor.f 186"
- WaitAtBarrier "opt33" 0.33 "ssor.f 207"
- WaitAtBarrier "opt34" 0.01 "ssor.f 207"
Application Test: APART Test Suite (ATS)

- ATS: Collection of simple applications, each demonstrating a specific situation of inefficient execution
  - Developed for testing automated and manual performance analysis tools
  - Many cases are easily detected by Periscope

- Example: “Imbalance at Barrier”
  - Un-Balanced workload of threads prior to a (user-added) OpenMP barrier

  Property 'WaitAtBarrier'
  holds for 'imbalance_at_barrier.c BARRIER [17]' on 'opt02'
  severity 0.359103, confidence 1.0

- Example: “Critical Section Contention”
  - Several threads try to enter a critical section concurrently

  Property 'CriticalSectionContention'
  holds for 'critical_section_contention.c CRITICAL [15--16]' on 'opt02'
  severity 0.374319, confidence 1.0

  Property 'ImbalanceInParallelRegion'
  holds for 'critical_section_contention.c PARALLEL [11--19]' on 'opt02'
  severity 0.374954, confidence 1.0
Test: NAS Parallel Benchmarks (OpenMP)

- NAS Class „C“ Benchmarks
  - 8 Applications: BT, CG, EP, FT, IS, LU, MG, SP
  - Batch mode, SGI Alitx ccNUMA machine, 1.6 GHz Itanium II processors
  - 8 Threads on 8 Processors, Periscope node-level agent ran on the 9th processor
  - Search for properties initiated at program termination

- Number of properties found:

<table>
<thead>
<tr>
<th>Property</th>
<th>BT</th>
<th>CG</th>
<th>EP</th>
<th>FT</th>
<th>IS</th>
<th>LU</th>
<th>MG</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitAtBarrier</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ImbalanceInParallelSections</td>
<td></td>
<td>12</td>
<td>13</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>ImbalanceInParallelLoop</td>
<td></td>
<td>12</td>
<td>13</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>ImbalanceInParallelRegion</td>
<td></td>
<td>6</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>UnparallelizedInSingle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>UnparallelizedInMaster</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImbalanceDueToNotEnoughSections</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>ImbalanceDueToUnevenSectionDistribution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CriticalSectionContestion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LockContention</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Test: NAS Parallel Benchmarks (OpenMP)

- The three most severe properties discovered:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Property</th>
<th>Region</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>ImbalanceInParallelLoop</td>
<td>rhs.f 177--290</td>
<td>0.0446</td>
</tr>
<tr>
<td>BT</td>
<td>ImbalanceInParallelLoop</td>
<td>y.solve.f 40--394</td>
<td>0.0353</td>
</tr>
<tr>
<td>BT</td>
<td>ImbalanceInParallelLoop</td>
<td>rhs.f 299--351</td>
<td>0.0347</td>
</tr>
<tr>
<td>CG</td>
<td>ImbalanceInParallelLoop</td>
<td>cg.f 556--564</td>
<td>0.0345</td>
</tr>
<tr>
<td>CG</td>
<td>ImbalanceInParallelRegion</td>
<td>cg.f 772--795</td>
<td>0.0052</td>
</tr>
<tr>
<td>CG</td>
<td>ImbalanceInParallelRegion</td>
<td>cg.f 883--957</td>
<td>0.0038</td>
</tr>
<tr>
<td>EP</td>
<td>ImbalanceInParallelRegion</td>
<td>ep.f 170--230</td>
<td>0.0078</td>
</tr>
<tr>
<td>EP</td>
<td>ImbalanceInParallelLoop</td>
<td>ep.f 129--133</td>
<td>0.0001</td>
</tr>
<tr>
<td>FT</td>
<td>ImbalanceInParallelLoop</td>
<td>ft.f 606--625</td>
<td>0.0676</td>
</tr>
<tr>
<td>FT</td>
<td>ImbalanceInParallelLoop</td>
<td>ft.f 653--672</td>
<td>0.0304</td>
</tr>
<tr>
<td>FT</td>
<td>ImbalanceInParallelLoop</td>
<td>ft.f 227--235</td>
<td>0.0269</td>
</tr>
<tr>
<td>IS</td>
<td>WaitAtBarrier</td>
<td>is.c 526</td>
<td>0.0272</td>
</tr>
<tr>
<td>IS</td>
<td>ImbalanceInParallelRegion</td>
<td>is.c 761--785</td>
<td>0.0087</td>
</tr>
<tr>
<td>IS</td>
<td>ImbalanceInParallelLoop</td>
<td>is.c 397--403</td>
<td>0.0020</td>
</tr>
<tr>
<td>LU</td>
<td>WaitAtBarrier</td>
<td>ssor.f 211</td>
<td>0.0040</td>
</tr>
<tr>
<td>LU</td>
<td>WaitAtBarrier</td>
<td>ssor.f 182</td>
<td>0.0032</td>
</tr>
<tr>
<td>LU</td>
<td>ImbalanceInParallelLoop</td>
<td>rhs.f 189--309</td>
<td>0.0011</td>
</tr>
<tr>
<td>MG</td>
<td>ImbalanceInParallelLoop</td>
<td>mg.f 608--631</td>
<td>0.0831</td>
</tr>
<tr>
<td>MG</td>
<td>ImbalanceInParallelLoop</td>
<td>mg.f 779--815</td>
<td>0.0291</td>
</tr>
<tr>
<td>MG</td>
<td>ImbalanceInParallelLoop</td>
<td>mg.f 536--559</td>
<td>0.0248</td>
</tr>
<tr>
<td>SP</td>
<td>ImbalanceInParallelLoop</td>
<td>x.solve.f 27--296</td>
<td>0.0285</td>
</tr>
<tr>
<td>SP</td>
<td>ImbalanceInParallelLoop</td>
<td>y.solve.f 27--292</td>
<td>0.0265</td>
</tr>
<tr>
<td>SP</td>
<td>ImbalanceInParallelLoop</td>
<td>z.solve.f 31--326</td>
<td>0.0239</td>
</tr>
</tbody>
</table>
Test: NAS Parallel Benchmarks (MPI Version)

- NAS CG Benchmark
  - 64 MPI processes, AMD Opteron, Infiniband Cluster, 16 Nodes

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Location</th>
<th>Processes</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointToPointComm0vhd</td>
<td>MPI_SEND</td>
<td>26</td>
<td>0.350885</td>
</tr>
<tr>
<td>PointToPointComm0vhd</td>
<td>MPI_SEND</td>
<td>30</td>
<td>0.285780</td>
</tr>
<tr>
<td>PointToPointComm0vhd</td>
<td>MPI_SEND</td>
<td>8</td>
<td>0.036432</td>
</tr>
<tr>
<td>WaitAtMpiBarrier</td>
<td>MPI_BARRIER</td>
<td>all</td>
<td>0.000126</td>
</tr>
<tr>
<td>PointToPointComm0vhd</td>
<td>MPI_RECV</td>
<td>all</td>
<td>0.000157</td>
</tr>
<tr>
<td>CollectiveComm0vhd</td>
<td>MPI_REDUCE</td>
<td>all</td>
<td>3.276e-06</td>
</tr>
</tbody>
</table>
Summary

- High level of abstraction of performance analysis results
  - **Kind** of inefficiency (by virtue of the name of the property)
  - **Severity** (negative impact)
  - **Location** (program, thread, program region)

- Scalability of the analysis
  - Size of the tool can be adapted to the size of the machine/application
  - Bigger machine => more agents
  - Efficient collection of results using tree-structure

- Extensibility of the tool
  - New properties can be loaded dynamically (in shared objects)
  - Extension of the performance analysis knowledge without changing the core of the tool

- Demo
  - At LRZ (Leibniz Computing Center) Booth, Tuesday and Wednesday at 11:00

Thank you for your attention!