Modular Infrastructure for Automation of Performance Optimization

Jim Browne, Leo Fialho, Ashay Rane

http://www.tacc.utexas.edu/perfexpert

THE UNIVERSITY OF TEXAS AT AUSTIN
Contents

1. Motivation
2. Introduction
3. Workflow and Implementation
4. Application Example
5. Modules and Implementation
6. Status
7. Conclusions
Motivation: why PerfExpert?

Problem: HPC systems operate far below peak
- Chip/node architectural complexity is growing rapidly
- Performance optimization for these chips requires deep knowledge of architectures, code patterns, compilers, etc.

Performance optimization tools
- Powerful in the hands of experts
- Require detailed performance and system expertise
- HPC application developers are domain experts, not computer gurus

Result: Many HPC programmers do not use these tools (seriously)
Goal for PerfExpert: democratize optimization!

Subgoals:
- Make use of the tool as simple as possible
- Start with only chip/node level optimization
- Make it adaptable across multiple architectures
- Design for extension to communication and I/O performance

How to accomplish?
- Formulate the performance optimization task as a workflow of subtasks
- Leverage the state-of-the-art: build on the best available tools for the subtasks to minimize the effort and cost of development
- Automate the entire workflow
**Introduction**

The four stages of automatic performance optimization:

1. Measurement and attribution
2. Analysis, diagnosis and identification of bottlenecks
3. Selection of effective optimizations
4. Implementation of optimizations

**Use of State-of-the-Art:**

1. HPCToolkit, Intel VTune, Score-P, **MACPO**
2. **PerfExpert Team**
3. **PerfExpert Team** based on ROSE, PIPS, Bison and Flex
Uniqueness of PerfExpert:

- Workflow is now fully implemented complete with automated optimizations
- Framework for implementing optimizations complete and several optimizations completed
- Integrates both code segment focused and data structure based measurements (MACPO)
- Workflow will apply to communication and I/O optimization as well
Workflow

Instrumentation
- Module 1
- Module 2
- Module ...
- Module N

Compilation
- Module 1
- Module 2
- Module ...
- Module N

Database

original source code

optimized source code

binary object

Module 1
Module 2
Module ...
Module N

POIT 1
POIT 2
POIT ...
POIT N

Optimization

Analysis

User Input

Workflow Tool

Relationship to Other Research
- Optimally guided autotuning
- Unification of compile-time and runtime optimization
Key Points

- The workflow is a sequence of steps through phases (e.g., compilation, measurement, analysis, recommendations, optimization, ...)
- Workflow cycles through its steps until the system has exhausted the optimizations it can implement.
- Workflow concludes by giving user recommendations for other possible optimizations.
- Functionality of modules can be changed (e.g., measure energy consumption instead of performance).
- Workflow management is a C program which can be modified.
Modular Infrastructure

Key Points

- Modules are dynamic loadable components (shared libraries) which implement one or more phases.
- Modules require, interact, and coordinate with other modules.
- The order each module/phase is executed (which defines the workflow) is defined by the pre-requisites of each module and the interaction between them.
- Modules may interact with other modules (e.g., one module passing arguments to another module).
- Each module defines its outputs (a table on the database) and interprets its inputs (from the tool or from a table in the database).
Workflow

Key Points

- The workflow is a sequence of steps through phases (e.g., compilation, measurement, analysis, recommendations, optimization, ...)
- Workflow cycles through its steps until the system has exhausted the optimizations it can implement
- Workflow concludes by giving user recommendations for other possible optimizations
- Functionality of modules can be changed (e.g., measure energy consumption instead of performance)
- Workflow management is a C program which can be modified
Workflow

Instrumentation

Compilation

binary object

Measurements

User Input

Workflow Tool

original source code

Database

optimized source code

Optimization

Analysis

Relationship to Other Research

- Optimally guided autotuning
- Unification of compile-time and runtime optimization
Modular Infrastructure

**Instrumentation phase modules...**
- should be able to instrument the source or binary code
- are generally implemented by modules which also implement the measurement phase
- may accept parameters (such as which functions would be instrumented)

**Compilation phase modules...**
- should be able to compile source codes (one or multiple languages, one or multiple architectures) and produce an executable binary
- may accept compiler flags and libraries to link with the final binary code
Modular Infrastructure

**Instrumentation phase modules...**
- should be able to instrument the source or binary code
- are generally implemented by modules which also implement the measurement phase
- may accept parameters (such as which functions would be instrumented)

**Compilation phase modules...**
- should be able to compile source codes (one or multiple languages, one or multiple architectures) and produce an executable binary
- may accept compiler flags and libraries to link with the final binary code
Measurement phase modules...

- collect specific data (memory access pattern, MPI communication, OpenMP runtime info, hardware performance counters, ...) from the application execution
- may accept compiler flags and libraries to link with the final binary code
- may use existing tools (HPCToolkit, VTune, Score-P, etc.)
- store collected data into the database, making them available for all optimization steps including other measurement/analysis modules
Modular Infrastructure

Analysis phase modules...

- map measurements to meaningful metrics
- may generate performance analysis reports to the user
- may use existing tools (MACPO, Scalasca, etc.)
- store compiled data into the database, making them available for all optimization steps including other measurement/analysis modules
Optimization phase modules... are not modules!

- Each optimization is implemented by a separate Performance Optimization Implementation Tool – POIT.
- Implementing a recommendation may include: adding compiler flags, changing the source code, setting an execution environment variable, ...
- Some POITs are implement using the ROSE infrastructure or PIPS (a scripting language tool to express code transformations), but may the implemented using anything.
- Each POIT has to validate the feasibility of applying the recommendation (data dependency, code pattern, environment setup, ...) before applying it.
Example: Loop Interchange/Tiling for Heat Transfer Code

Original Source Code

DO K = SZ1, EZ1
  DO I = SX1, EX1
    DO J = SY1, EY1
      D(I,J,K) = Stem(I,J,K) + A.E(I,J,K) * P.OLD(I+1,J,K) &
                   + A.W(I,J,K) * P.OLD(I-1,J,K) + A.T(I,J,K) * P.OLD(I,J,K+1) &
                   + A.B(I,J,K) * P.OLD(I,J,K-1)
    ENDDO
  ENDDO
DO I = EX1, SX1, -1
  DO J = EY1, SY1, -1
    P(I,J,K) = TDMA.P(I,J,K) * P(I,J+1,K) + TDMA.Q(I,J,K)
  ENDDO
ENDDO
ENDDO
ENDDO
### Example: Loop Interchange/Tiling for Heat Transfer Code

#### Performance Analysis

Loop in function tdma in tdma.f90:51 (74.77% of the total runtime)

<table>
<thead>
<tr>
<th>Instructions Ratio</th>
<th>%</th>
<th>0</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>- floating point</td>
<td>8.4</td>
<td>&gt;&gt;&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- data accesses</td>
<td>0.0</td>
<td>[</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*GFLOPS (% max)</td>
<td>5.8</td>
<td>&gt;&gt;&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- scalar</td>
<td>0.2</td>
<td>[</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- packed</td>
<td>5.6</td>
<td>&gt;&gt;&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Performance Assessment

<table>
<thead>
<tr>
<th>Performance Assessment</th>
<th>LCPI</th>
<th>good</th>
<th>okay</th>
<th>fair</th>
<th>poor</th>
<th>bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>*overall</td>
<td>1.35</td>
<td>&gt;&gt;&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Slowdown Caused By

<table>
<thead>
<tr>
<th>Slowdown Caused By</th>
<th>LCPI</th>
<th>(interpretation varies according to the metric)</th>
</tr>
</thead>
<tbody>
<tr>
<td>*data accesses</td>
<td>2.23</td>
<td>&gt;&gt;&gt;</td>
</tr>
<tr>
<td>- L1 cache hits</td>
<td>0.02</td>
<td>[</td>
</tr>
<tr>
<td>- L2 cache hits</td>
<td>0.32</td>
<td>&gt;&gt;&gt;</td>
</tr>
<tr>
<td>- L3 cache hits</td>
<td>0.22</td>
<td>&gt;&gt;&gt;</td>
</tr>
<tr>
<td>- LLC misses (memory)</td>
<td>1.67</td>
<td>&gt;&gt;&gt;</td>
</tr>
<tr>
<td>*instruction accesses</td>
<td>0.01</td>
<td>[</td>
</tr>
<tr>
<td>- L1 hits</td>
<td>0.00</td>
<td>[</td>
</tr>
<tr>
<td>- L2 hits</td>
<td>0.00</td>
<td>[</td>
</tr>
<tr>
<td>- L2 misses</td>
<td>0.01</td>
<td>[</td>
</tr>
<tr>
<td>*data TLB</td>
<td>0.00</td>
<td>[</td>
</tr>
<tr>
<td>*instruction TLB</td>
<td>0.00</td>
<td>[</td>
</tr>
<tr>
<td>*branch instructions</td>
<td>0.02</td>
<td>[</td>
</tr>
<tr>
<td>- correctly predicted</td>
<td>0.01</td>
<td>[</td>
</tr>
<tr>
<td>- mispredicted</td>
<td>0.01</td>
<td>[</td>
</tr>
<tr>
<td>*FP instructions</td>
<td>2.15</td>
<td>&gt;&gt;&gt;</td>
</tr>
<tr>
<td>- fast FP instructions</td>
<td>2.15</td>
<td>&gt;&gt;&gt;</td>
</tr>
<tr>
<td>- slow FP instructions</td>
<td>0.00</td>
<td>[</td>
</tr>
</tbody>
</table>
Example: Loop Interchange/Tiling for Heat Transfer Code

Original Source Code

```plaintext
DO K = SZ1, EZ1
  DO I = SX1, EX1
    DO J = SY1, EY1
      D(I,J,K)=Term(I,J,K)+A_E(I,J,K)*P_OLD(I+1,J,K) &
       +A_W(I,J,K)*P_OLD(I-1,J,K)+A_T(I,J,K)*P_OLD(I,J,K+1) &
       +A_B(I,J,K)*P_OLD(I,J,K-1)
       *TDMA_P(I,J-1,K))
      TDMA_Q(I,J,K)=(D(I,J,K)+A_S(I,J,K)*TDMA_Q(I,J-1,K))
       /(A_P(I,J,K)-A_S(I,J,K)*TDMA_P(I,J-1,K))
    ENDDO
  ENDDO
DO I = EX1, SX1, -1
  DO J = EY1, SY1, -1
    P(I,J,K)=TDMA_P(I,J,K)*P(I,J+1,K)+TDMA_Q(I,J,K)
  ENDDO
ENDDO
ENDDO
ENDDO
```
**Example: Loop Interchange/Tiling for Heat Transfer Code**

### Performance Analysis

Loop in function *tdms* in *tdms.f90:*1 (74.77% of the total runtime)

<table>
<thead>
<tr>
<th>Instructions Ratio</th>
<th>%</th>
<th>0</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>- floating point</td>
<td>8.4</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- data accesses</td>
<td>0.0</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- GFLOPS (% max)</td>
<td>5.9</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- scalar</td>
<td>0.2</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- packed</td>
<td>5.6</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
</tbody>
</table>

**Performance Assessment**

<table>
<thead>
<tr>
<th>LCPI</th>
<th>good</th>
<th>okay</th>
<th>fair</th>
<th>poor</th>
<th>bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>overall</td>
<td>1.35</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
</tbody>
</table>

**Slowdown Caused By**

<table>
<thead>
<tr>
<th>LCPI (interpretation varies according to the metric)</th>
<th>[   ]</th>
<th>[   ]</th>
<th>[   ]</th>
<th>[   ]</th>
<th>[   ]</th>
<th>[   ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>- data accesses</td>
<td>2.23</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- L1 cache hits</td>
<td>0.02</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- L2 cache hits</td>
<td>0.32</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- L3 cache hits</td>
<td>0.22</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- LLC misses (memory)</td>
<td>1.67</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- instruction accesses</td>
<td>0.01</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- L1 hits</td>
<td>0.00</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- L2 hits</td>
<td>0.00</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- L2 misses</td>
<td>0.01</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- data TLB</td>
<td>0.00</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- instruction TLB</td>
<td>0.00</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- branch instructions</td>
<td>0.02</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- correctly predicted</td>
<td>0.01</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- mispredicted</td>
<td>0.01</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- FP instructions</td>
<td>2.15</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- fast FP instructions</td>
<td>2.15</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
<tr>
<td>- slow FP instructions</td>
<td>0.00</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
<td>[   ]</td>
</tr>
</tbody>
</table>
**Modules and Implementation**

**gcc, g++, gfortran, icc, icpc, and ifort modules...**
- Implement only the compilation phase
- Interact with the GNU and Intel compilers
- Accept parameters (compilation options and libraries to link)

**Make module...**
- Implement only the compilation phase
- It is a simple wrap around the well known GNU make utility
- Accept parameters (depend on the actual Makefile to support those parameters)
<table>
<thead>
<tr>
<th><strong>LCPI – Local Cycles per Instruction module...</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>implements the measure and analysis phases</td>
</tr>
<tr>
<td>does not perform any measure at all but during the execution of its measurement phase it sets the performance counter other modules (such as HPCToolkit or VTune) should take into consideration</td>
</tr>
<tr>
<td>compile metrics according to six categories: data accesses, instruction accesses, data TLB, instruction TLB, branch instructions, and floating-point instructions</td>
</tr>
<tr>
<td>identifies the bottleneck sources using six categories</td>
</tr>
<tr>
<td>stores the computed metrics into the database</td>
</tr>
<tr>
<td>reports analysis for each performance bottleneck found</td>
</tr>
</tbody>
</table>
Thread Imbalance (preliminary version) module...

- implements the measure and analysis phases
- does not perform any measure at all but, as the LCPI module, interfaces with other modules to set the performance counter it needs
- is currently based on performance event counters
- reports imbalance analysis for each thread
- future versions will have runtime system integration (OMPT and/or Score-P/OPARI)
SQL Rules module...

- implements only the recommendation phase
- based on the metrics provided by the LCPI analysis
- maps the LCPI six different categories to recommendations
- includes a ranking strategy to select the recommendation which may provide the best benefit
- designed as a rule-based strategy combining knowledge from multiple domains
- rules are expressed as a SQL query
- support multiple rules simultaneously
Status

- Use as the primary tool for performance optimization on TACC machines
- The Flemish Supercomputer Center (VSC) adopted PerfExpert as a pre-requirement to use their machines
- Downloaded and installed by over 65 institutions in over 20 different countries
- Available at: https://www.tacc.utexas.edu/perfexpert
Future Work

- Better integration of data access metrics
- Increase the number of recommendations and POITs [continuous task]
- New algorithms/modules for selection of recommendations [continuous task]
- Support for MIC architecture [done!]
- Support for OpenMP- and MPI-related recommendations
- Support for OpenMP- and MPI-related optimizations
- Support for GPU chips and heterogeneous architectures
- Data intensive applications (I/O instrumentation, measurement, analysis, recommendation, and optimization)