Using Graph-Based Characterization for Predictive Modeling

Assistant Prof. John Cavazos
University of Delaware
Department of Computer and Information Sciences
PLEASE HELP
I'm hungry for code
Can significantly Intel’s compiler by several factors

[Work done with Eunjung Park and Marco Alvarez and published at International Symposium on Code Generation and Optimization, 2012]
Finding the Best Optimization is Difficult

Applied Optimization Combinations Sorted by Increasing Actual Speedup

2MM (Matrix Multiply)

Normalized Speedup to ICC-fast

Actual Speedup

Normalized to baseline

PoCC

Fusion

Unrolling

Tiling

Parallelization

Vectorization

600 opt combinations

Optimized versions
Finding the Best Optimization is Difficult

Normalized Speedup to ICC-fast

Applied Optimization Combinations Sorted by Increasing Actual Speedup

PoCC

- Fusion
- Unrolling
- Tiling
- Parallelization
- Vectorization

Optimized versions

600 opt combinations

2MM (Matrix Multiply)

: Degrade
: Improvement > 1.0x
: Significant Improvement >= 6.0X
: The best Improvement

Actual Speedup

Normalized Speedup to baseline

Motivation
Finding the Best Optimization is Difficult

State-of-the-art

Graph-based

Build & Use

Evaluation
Observations

☐ Highest-level opt flag (-fast) does not give best perf

☐ Low density of “very” good points in the optimization space

☐ No single optimization configuration best for all programs

☐ Bundled flags are doomed to fail
Using a Prediction Model

Model follows the trend of actual speedup

Applied Optimization Combinations Sorted by Increasing Actual Speedup

Normalized Speedup to-fast in Intel ICC Compiler

Actual Speedup

Model

PoCC

Fusion
Unrolling
Tiling
Parallelization
Vectorization

Optimized versions

2MM (Matrix Multiply)

600 opt combinations
Using a Prediction Model

Normalized Speedup to -fast
in Intel ICC Compiler

Applied Optimization Combinations Sorted by Increasing Actual Speedup

PoCC
- Fusion
- Unrolling
- Tiling
- Parallelization
- Vectorization

2MM (Matrix Multiply)

: top 1 - 5 Predicted Combinations
: top 6 - 10 Predicted Combinations
State-of-the-art Characterizations

### Dynamic

#### Performance Counters

<table>
<thead>
<tr>
<th>Category</th>
<th>List of PCs</th>
<th>Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Line Access</td>
<td>CA_CLN, CA-ITV, ..</td>
<td>3</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>L1_DCA, L1_DCH, ..</td>
<td>10</td>
</tr>
<tr>
<td>L2/L3 Cache</td>
<td>L2_DCA, L2_DCM, ..</td>
<td>16</td>
</tr>
<tr>
<td>Branch Related</td>
<td>BR_TKN, BR_NTK, ..</td>
<td>7</td>
</tr>
<tr>
<td>Floating Point</td>
<td>FDV_INS, FP_OPS, ..</td>
<td>6</td>
</tr>
<tr>
<td>Interrupt/Stall</td>
<td>HW_INT, RES_STL</td>
<td>2</td>
</tr>
<tr>
<td>TLB</td>
<td>TLB_DM, TLM_IM, ..</td>
<td>4</td>
</tr>
<tr>
<td>Total Cycle/Inst.</td>
<td>TOT_CYC, TOT_IIS, ..</td>
<td>3</td>
</tr>
<tr>
<td>Load/Store Inst.</td>
<td>LD_INS, SR_INS</td>
<td>2</td>
</tr>
<tr>
<td>SIMD Inst</td>
<td>VEC_INS, VEC_DP, ..</td>
<td>3</td>
</tr>
</tbody>
</table>

### Static

#### Performance Counters

1: N-1 Programs

#### Underlying Architecture

1: N-1 Programs

Complied with -O0

Backend Compiler

N-1: Performance Counters for N-1 Programs
State-of-the-art Characterizations

Dynamic

Static

Source Code

N-1 programs

Milepost GCC

1

... 

R

Source code features for N-1 programs

Performance

Counters

Reactions

Collect
represent

N

Source

Use
machine

GCC

... 

1

... 

N-1

1

... 

R

...
State-of-the-art Characterizations

- **Dynamic**
  - Performance Counters
  - Reactions

- **Static**
  - Source Code

**Advantages**
- Easy to collect and works with standard ML algorithms

**Disadvantages**
- Fixed-length vector loses information about program (e.g., Lost information about loop structure)
Graph-based Characterization

**Dynamic**
- Performance
- Counters

**Static**
- Reactions
- Source Code

**Graph-based**

Graph-based - CFG

N-1 programs

- MinIR

CFG topology and feature of each node for N-1 programs

Feature vector of each bb

- bb1
- bb6

CFG Topology

- bb1 -> bb2, bb3
- bb2 -> bb5
- ... -> bb5
- bb5 -> bb6
Graph-based Characterization using CFG

2MM - Matrix Multiplication

Feature vector for each node
1: <1,0,0,0,0,0,0,0,1,0,0,0>
2: <4,0,0,0,0,1,0,0,1,0,2>
3: <16,2,0,1,0,3,1,0,0,1,0,8>
...
16: <2,1,0,0,0,0,0,0,1,0,0,0>
17: <3,0,0,0,0,0,1,1,0,1,0,0>
18: <1,0,0,0,0,0,0,0,1,0,0,0>

Topology of CFG
1->6
2->4
3->4
...
16->17
17->10
17->18

Features of Each Node
Number of Instructions
Number of Add Instruction
Number of Sub Instruction
Number of Mult Instruction
Number of Div Instruction
Number of Load Instruction
Number of Store Instruction
Number of Comparisons
Number of Conditional Branches
Number of Unconditional Branches
Number of Phi Nodes
Number of GetElementPtr Instruction
Building the Prediction Model

Control Flow Graph Features

Optimization Combination $O$

Predicted speedup of $O$ over the Baseline
Building the Prediction Model

CFG features for N-1 programs

Opt. Combinations and their speedup over baseline for N-1 programs

Machine Learning Algorithm

Generated model for a given machine
Evaluating Predictors

- $N$ Number of evaluations = $N$-Shot model
- Non-iterative scenario (1-shot model)
- Iterative scenario
  - 5-shot model

Sort by Predicted Speedup

**top 5**

Predicted speedup for each combination

CFG Features

Optimization Combinations
Case Study

- PoCC (Source-to-source Polyhedral Compiler)
- Since source-to-source, requires a backend compiler

All Optimizations Considered
Unrolling Factor = 0, 2, 4, and 8
Loop Fusion = nofuse, maxfuse, smartfuse
Loop tiling = 1, 32 (outer, middle, inner loops)
Parallelization = on/off
Vectorization = on/off
Experimental Configuration

- Hardware Configuration
  - (Nehalem) Intel Xeon E5620 2.4GHz, 16 H/W threads
  - (Quad) Intel Core2 Quad Q9650 3.0GHz, 4 H/W threads

- Software Configuration
  - Backend compiler: ICC (Baseline: -fast) GCC (Baseline -O3)
  - Machine Learning: Support Vector Machines
  - Benchmarks: PolyBench V2.1 (30 scientific kernels)
    - 26 FP, 4 INT C kernels with Average LOC 337
    - Linear Algebra, Solvers, Stencil Comps, Data Mining
## Experiment: CFG vs. State-of-the-art

- **Single-Shot SVM Model**

<table>
<thead>
<tr>
<th></th>
<th>Dynamic</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PC</td>
<td>SRC</td>
<td>CFG</td>
<td>OPT</td>
</tr>
<tr>
<td>Nehalem-GCC</td>
<td>4.6x (51%)</td>
<td>5.2x (57%)</td>
<td>5.7x (63%)</td>
<td>9.06x</td>
</tr>
<tr>
<td>Nehalem-ICC</td>
<td>2.5x (45%)</td>
<td>2.7x (48%)</td>
<td>3.0x (53%)</td>
<td>5.60x</td>
</tr>
<tr>
<td>Quad-GCC</td>
<td>2.0x (54%)</td>
<td>2.2x (59%)</td>
<td>2.7x (73%)</td>
<td>3.67x</td>
</tr>
<tr>
<td>Quad-ICC</td>
<td>1.7x (37%)</td>
<td>1.8x (40%)</td>
<td>2.8x (60%)</td>
<td>4.57x</td>
</tr>
</tbody>
</table>

We achieved significant performance improvements by using single-shot CFG model (static technique)!

**Note:** ICC -par performs poorly (1.8x on Nehalem and 1.3x on Quad)
Experiment: CFG vs. State-of-the-art

- 5-Shot SVM Model

<table>
<thead>
<tr>
<th></th>
<th>Dynamic</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PC</td>
<td>SRC</td>
</tr>
<tr>
<td>Nehalem-GCC</td>
<td>5.2x (57%)</td>
<td>5.8x (63%)</td>
</tr>
<tr>
<td>Nehalem-ICC</td>
<td>3.3x (58%)</td>
<td>2.9x (52%)</td>
</tr>
<tr>
<td>Quad-GCC</td>
<td>2.4x (66%)</td>
<td>2.4x (65%)</td>
</tr>
<tr>
<td>Quad-ICC</td>
<td>2.6x (57%)</td>
<td>2.5x (55%)</td>
</tr>
</tbody>
</table>

We achieved up to 88% of OPT just in 5 iterations!
## Optimization Combination Analysis: Quad-ICC

<table>
<thead>
<tr>
<th></th>
<th>Fusion and Distribution</th>
<th>Tiling</th>
<th>pluto-parallel</th>
<th>auto-parallel</th>
<th>pluto-prevec</th>
<th>vectorization</th>
<th>Loop-unroll-and-jam</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>smart</td>
<td>no</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>2mm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3mm</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>adi</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>atax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bicg</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>cholesky</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>covariance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>doigtgen</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>durbin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>dynprog</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ftdtd-2d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ftdtd-apml</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gauss-filter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>gemm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>gemver</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>gesummvm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>gramschmidt</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>jacobi-1d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>jacobi-2d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>lu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ludcmp</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>mvt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>reg_detect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>seidel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>symm</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>syr2k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>syrk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>trisolv</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>trmm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
A Closer Look

- ATAX and BICG perform best using same optimization sequence.
- ATAX - matrix transpose and vector multiplication.
- BiCG - Kernel from the Bi-conjugate gradient solver.
- No Fusion, No Unrolling, Wavefront Parallelism, Parallelizing the Outer Loop, and Tiling the Inner Loop.
- Graph-based technique predicts good sequences to apply for both these kernels.
compiler's

Never send a human to do a machine's job.
Using Graph-Based Characterization for Predictive Modeling

Thank You!

Assistant Prof. John Cavazos
cavazos@udel.edu
http://www.cis.udel.edu/~cavazos