Automatic Creation of Tile Size Selection Models

Tomofumi Yuki
Lakshminarayanan Renganarayanan
Sanjay Rajopadhye
Charles Anderson
Alexandre Eichenberger
Kevin O'Brien

Colorado State University    IBM Research
Tile Size Selection Problem

- Tiling is an optimization with a parameter “tile size”
- Finding good tile sizes is essential to benefit from tiling
- Good tile sizes can be different for each hardware/application
Problems

- Several factors influence performance of tiled code
- Hardware and software keep changing
- Analytical Models (existing approach):
  - Require expert knowledge and significant time
- Auto Tuning/Iterative Compilation:
  - Long compilation time

Can we automate TSS model development?
Problems

● Several factors influence performance of tiled code
● Hardware and software keep changing
● Analytical Models (existing approach):
  ● Require expert knowledge and significant time
● Auto Tuning/Iterative Compilation:
  ● Long compilation time

Can we automate TSS model development?  
YES we use ML to automate this process
Outline

- Background
  - Tiling
  - Performance considerations for tiled codes
  - Neural Networks
- Approach
- Performance Evaluation
- Conclusions and Future Work
Tiling

original loop
for (i=0; i<=8; i++)
  for (j=0; j<=8; j++)

tiled loop
for (ti=0; ti <= 8; ti+=3)
  for (tj=0; tj <= 8; tj+=3)
    for (i=ti; i < ti+3; i++)
      for (j=tj; j < tj+3; j++)
original loop
for (i=0; i<=8; i++)
  for (j=0; j<=8; j++)

tiled loop
for (ti=0; ti <= 8; ti+=3)
  for (tj=0; tj <= 8; tj+=3)
    for (i=ti; i < ti+3; i++)
      for (j=tj; j < tj+3; j++)
Tiling for Locality

- Array M is indexed by j
  Untiled: 9 locations accessed before next i
  Tiled: 3 locations accessed before next i
  => Better reuse if cache cannot store 9 elements
Performance Considerations

- **Different Types of Cache Misses**
  - **Cold Miss**
    - Unavoidable cost when data is first read into cache
  - **Capacity Miss**
    - Evicted from cache before reuse due to capacity
    - LRU eviction is assumed
  - **Conflict Miss**
    - Evicted from cache before reuse due to conflicts
    - Self conflict and cross conflict
Hardware Prefetching

- Hardware to detect access patterns and load data ahead of time
- Large impact on performance of tiled code
Hardware Prefetching

- Hardware to detect access patterns and load data ahead of time
- Large impact on performance of tiled code

Unit-Stride prefetching: \( \text{next} = \text{prev} + 1 \)
Neural Networks

Important Characteristics
- Supervised Learning:
  Requires input and desired output for training
- Using neural networks is fast (matrix-vector product)
- Many parameters (number of nodes, layers, and so on)
Outline

• Background
• Approach
  • Class of Programs
  • TSS Model Structure
  • Data Collection
  • Training
  • Use of the Model
• Performance Evaluation
• Conclusions and Future Work
Class of Programs

- **Affine Control Loops**
  - Tiled code generators are available
  - Many programs that benefit from tiling fit

- **Constraint on Tiling**
  - One-level tiling for cache locality
  - Cubic tile sizes
    - To limit data collection time
  - 2D data, 3D loops
    - 4D+ loops are handled by tiling innermost 3
TSS Model Structure

- **Input: Program Features**
  - High-level characterization of reuse
  - Total of 6 features
    - Based on number of references in the statement
      1. Prefetched
      2. Non-Prefetched
      3. Invariant
        - Each type is further separated by Read/Write

- **Output: Optimal Tile Size**
Overview of Our Approach

1. Data Collection

2. Learning TSS Models Using NN
   - One model for each architecture/compiler

3. Use of the Model During Compilation
   - Extract program features
   - Compute NN output

Only step 3 is performed during compilation
Data Collection

- **Use of Synthetic Programs**
  - Select a range of program features
  - Generate programs that has the required feature
  - Run the programs to find optimal tile sizes

- **Advantages**
  - Comprehensive and rich training data set
    - Uniform coverage
    - Avoid multiple programs with same features
    - Easy to get a large set of training data
Model Learning and Use

• Model Learning
  • Neural network parameters are manually tuned
    – Only step in model creation that is not automated
    – After designing a general structure, small tuning was required for different architecture

• Use
  • Feature extraction is straightforward
  • Computing NN output is instantaneous
  • Use of the model is inexpensive
Performance Evaluation

- Evaluated by comparing the performance of predicted tiles and the actual optimal
- Trained separate models for each architecture-compiler combination
- 3 architectures, 2 compilers each

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Compilers</th>
<th>L1 Cache</th>
<th>HW Prefetcher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opteron</td>
<td>PSC, GCC</td>
<td>64KB 2-way</td>
<td>unit-stride</td>
</tr>
<tr>
<td>Power5</td>
<td>XLC, GCC</td>
<td>32KB 4-way</td>
<td>unit-stride</td>
</tr>
<tr>
<td>Core2Duo</td>
<td>ICC, GCC</td>
<td>32KB 8-way</td>
<td>constant-stride</td>
</tr>
</tbody>
</table>
Results

Execution time using trained models, normalized to the true optimal

- No worse than 20% slower compared to the true optimal
- Consistent across all architecture-compiler combinations
Performance of LRW

- Analytical model that predicts square tiles [LRW]
- Tailored to take HW prefetching into account

Conclusions & Future Work

● Conclusions

Reasonably accurate TSS models can be automatically constructed with “Semantic Features + Synthetic Programs + NN”

● Implemented in the IBM XLC

● Future Work

  • Extending class of programs
  • Automatic NN parameter tuning
  • Extract insight from the model
Questions?