Pruning Strategies in Adaptive Off-line Auto-tuning

Lu Li, Usman Dastgeer, Christoph Kessler
Linköping University
Agenda

1. Motivation
2. Approach
3. Results
4. Conclusion
Problem Background

- Application needs tuned for optimal performance
- Performance tuning is challenging
  - Heterogeneous processors
  - Configuration diversity
- Auto-tuning needed
  - Performance Portability
- Approach: Implementation selection for a multi-variant component
Component Interface and Implementations

Implementation variant
- Platforms (CPU, accelerator cores)
- Algorithms
- Tunable parameter settings
- Compiler transformations
- ...

Meta-data
- Dependencies
- Resource requirements
- Deployment descriptors
- Performance prediction models
- ...

Component Interface

Variant meta-data

Variant meta-data

Variant meta-data

Variant meta-data
Staged Composition

**Static composition:**
at deployment time, user-guided or off-line autotuning + static narrowing of set of candidates

**Dynamic composition:**
on-line tuning, selection of the expected best component

Contributed or generated variants with static performance predictions
Guiding Composition: Empirical Models

- Analytical model
- Empirical model
  - No or little understanding of the target architecture
  - Off-line Empirical Models: Feed sampling data (training examples) into a prediction model, e.g. SVM, C4.5
Off-line Empirical Models: Pros and Cons

- Avoid “cold start”
- Controllable tuning process
- Tuning overhead

“Cold start” Effect

Example from Dastgeer et al. (ParCo’2011) on SkePU/StarPU integration, Coulombic potential grid execution, with 3 successive executions
Zoom into the Cons

- A closer look at off-line overhead
  - Exhaustive execution is not feasible
  - Previous work: Stargazer (random sampling)
- Training examples (Sampling data) is vital for performance prediction models
Attack the Cons: Observations

- In the context of performance tuning: a concrete example (Matrix-matrix multiplication)
- Smart sampling by heuristics

Assume it wins in between?
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Adaptive Sampling

- Sample only vertices of a space
- Recursive decomposition (cutting) for open spaces, controlled by maximum depth, etc
Formalization: Runtime context (e.g. Input Size)

- **Context Property Value Space (PVS)** is a $n$-dimensional finite space
  - A run-time context instance consists of $n$ context property values maps to a point in the PVS
  - PVS has $2^n$ vertices (corner points)

- Closed and open subspaces
  - Closed subspace: a subspace where a variant wins on all vertices. It heuristically approximate “uninteresting” subspaces
  - Open subspace: otherwise

- Recursive space decomposition
In real life, the world is smoothly changing, instances close by most of the time have the same labels, and we need not worry about all possible labelings.

Ethem Alpaydın in “Introduction to Machine Learning, second edition”

Our Convexity Assumption

If all vertices of a subspace share the same winner, then the winner wins in all points of the subspace statistically.
Adaptive Sampling In the Big Picture

- Off-line: *adaptive sampling*, tree construction

- On-line (Run-time)
  - Load the tree
  - Prediction
    - Closed space: look up winner on one vertex
    - Open space: Euclidean-based predictor
Adaptive Sampling In the Big Picture

- **Off-line:** adaptive sampling, tree construction

- **On-line (Run-time)**
  - Load the tree
  - Prediction
    - Closed space: look up winner on one vertex
    - Open space: Euclidean-based predictor
Techniques for Adaptive Sampling

Light oversampling
- Sample one extra point in the middle
- Detect holes
- Small increase in overhead
- May increase prediction accuracy
Further Techniques Based on Adaptive Sampling

Thresholding

- Relative threshold: \( \frac{\text{abs}(v_i - v_{\text{min}})}{v_{\text{min}}} \leq \theta \)
- Stop splitting early
- Reduce training overhead
- May decrease prediction accuracy

\[
\begin{align*}
V_1 & \quad V_1 \\
V_2 & \quad V_1
\end{align*}
\]
Further Techniques Based on Adaptive Sampling

Thresholding
- Relative threshold: $\frac{|v_i - v_{min}|}{v_{min}} \leq \theta$
- Stop splitting early
- Reduce training overhead
- May decrease prediction accuracy

![Diagram showing adaptive sampling with thresholds $V_1$ and $V_2$.]
Further Techniques Based on Adaptive Sampling

**Implementation Pruning**
- Only winners of the vertices will involve in the future sampling of the subspace
- Reduce overhead remarkably if many implementation variants are available for an interface
- May lead to loss of optimization potential
Further Techniques Based on Adaptive Sampling

Implementation Pruning

- Only winners of the vertices will involve in the future sampling of the subspace
- Reduce overhead remarkably if many implementation variants are available for an interface
- May lead to loss of optimization potential

Try 10 variants for each vertex
Further Techniques Based on Adaptive Sampling

**Implementation Pruning**
- Only winners of the vertices will involve in the future sampling of the subspace
- Reduce overhead remarkably if many implementation variants are available for an interface
- May lead to loss of optimization potential
Further Techniques Based on Adaptive Sampling

**Implementation Pruning**
- Only winners of the vertices will involve in the future sampling of the subspace
- Reduce overhead remarkably if many implementation variants are available for an interface
- May lead to loss of optimization potential
Implementation Pruning

- Only winners of the vertices will involve in the future sampling of the subspace
- Reduce overhead remarkably if many implementation variants are available for an interface
- May lead to loss of optimization potential

Try 2 winner variants for each untested vertex
## Platform

<table>
<thead>
<tr>
<th>Name</th>
<th>CPU</th>
<th>GPU</th>
<th>OS</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>16 Intel(R) Xeon(R) CPU X5550 @ 2.67GHz</td>
<td>3 GPUs: two nVidia Tesla C2050 and one Tesla C1060</td>
<td>RHEL 5.6</td>
<td>gcc 4.1.2 and nvcc V0.2.1221</td>
</tr>
<tr>
<td>Fermi</td>
<td>8 Intel(R) Xeon(R) CPU E5520 @ 2.27GHz</td>
<td>two Tesla M2050 GPUs</td>
<td>3.2.1-2-ARCH</td>
<td>gcc 4.6.2 and nvcc V0.2.1221</td>
</tr>
</tbody>
</table>
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Feature modeling</th>
<th>Range</th>
<th>Implementation variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix-matrix multi-</td>
<td>row size, column size of first matrix, column size of second matrix</td>
<td>(30, 30, 30) to (300, 300, 300)</td>
<td>Sequential implementation and a variant by loop rearrangement, CUDA impl., BLAS impl., Pthread impl. and five of its variants from loop rearrangement</td>
</tr>
<tr>
<td>matrix multiplication (MM)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorting (ST)</td>
<td>array size; discretization of array values distribution (sampled number of inversions)</td>
<td>(1,0) to (10k,10)</td>
<td>bubble sort, insertion sort, merge sort, quick sort</td>
</tr>
<tr>
<td>Path finder (PF)</td>
<td>row; column</td>
<td>(1,1) to (10k,20k)</td>
<td>OpenMP implementation, CUDA implementation</td>
</tr>
<tr>
<td>Backpropagation (BP)</td>
<td>array size</td>
<td>(1k) to (100k)</td>
<td>OpenMP implementation, CUDA implementation</td>
</tr>
</tbody>
</table>
Prediction Accuracy (%) on Cora: Baseline adaptive off-line sampling

- Path finder (Rodinia)
  - Depth 3: 90%
  - Depth 2: 85%
  - Depth 1: 70%
  - Depth 0: 40%

- Back propagation (Rodinia)
  - Depth 3: 95%
  - Depth 2: 88%
  - Depth 1: 75%
  - Depth 0: 50%

- Sorting
  - Depth 3: 92%
  - Depth 2: 86%
  - Depth 1: 72%
  - Depth 0: 45%

- Matrix Multiplication
  - Depth 3: 94%
  - Depth 2: 89%
  - Depth 1: 78%
  - Depth 0: 55%
Prediction Accuracy (%) on Fermi: Base-line adaptive off-line sampling

- Path finder (Rodinia)
- Back propagation (Rodinia)
- Sorting
- Matrix Multiplication

Legend:
- Depth 4
- Depth 3
- Depth 2
- Depth 1
- Depth 0
Other Metrics

- Absolute runtime overhead: 4 - 23 $\mu$s
- Relative runtime overhead: 0.2%
- Average sampling rate: 0.053%
Test against Convexity Assumption

**BP**

**MM**

**PF**

**ST**

- **Accuracy(%)** on closed space.
- **Percentage(%)** on closed space.
Test against Convexity Assumption

Accuracy(%) on closed space.  Percentage(%) on closed space.
Test against Convexity Assumption

![Bar chart showing BP accuracy and percentage on closed space for different depths.](chart)

- **Accuracy(%) on closed space.**
- **Percentage(%) on closed space.**
Thresholding Effect: Training Time

BP: Depth 4

PF: Depth 1

MM: Depth 4

ST: Depth 4
Thresholding Effect: Prediction Accuracy

BP: Depth 4

PF: Depth 1

MM: Depth 4

ST: Depth 4
Thresholding Effect: Prediction Accuracy

BP: Depth 4

PF: Depth 1

MM: Depth 4

ST: Depth 4
Thresholding Effect: Prediction Accuracy

BP: Depth 4
Oversampling Effect: Prediction Accuracy

BP

PF

ST

MM

Oversampling Effect: Prediction Accuracy

Oversampling Effect: Prediction Accuracy

BP

- Adaptive Sampling
- Adaptive Sampling with light oversampling
Oversampling Effect: Training Time

Adaptive Sampling.
Adaptive Sampling with light oversampling.
Implementation Pruning Effect: Training Time

![Graphs showing the impact of implementation pruning on training time for different methods.](image)

- **BP**
- **MM**
- **PF**
- **ST**

- Blue bars: Adaptive Sampling.
- Red bars: Adaptive Sampling with impl pruning.
Implementation Pruning Effect: Prediction Accuracy

BP

PF

MM

ST

Adaptive Sampling

Adaptive Sampling with impl pruning.
Combo Effect: Training Time

Adaptive Sampling with the combo of oversampling and impl pruning.
Combo Effect: Prediction Accuracy

- **BP**
  - Adaptive Sampling.
  - Adaptive Sampling with combo of oversampling and impl pruning.

- **MM**

- **PF**

- **ST**
Adaptive Sampling vs Random Sampling

BP

PF

MM

ST

- Random Sampling
- Adaptive Sampling
- Adaptive Sampling with the combo of oversampling and impl pruning.
Adaptive Sampling vs Random Sampling

**BP**

**MM**

**PF**

**ST**

Random Sampling  Adaptive Sampling  Adaptive Sampling with the combo of oversampling and impl pruning.
Adaptive Sampling vs Random Sampling

Random Sampling
Adaptive Sampling
Adaptive Sampling with the combo of oversampling and impl pruning.
Conclusion

- Convexity Assumption holds for all benchmarks used, more to test
- Three techniques help to decrease training time or increase prediction accuracy
  - Thresholding
  - Light Oversampling
  - Implementation pruning
- The right combination can combine the advantages
- Adaptive sampling shows benefits against random sampling

More questions please contact me: lu.li@liu.se, +46704759692