MACHINE LEARNING FOR PROGRAM OPTIMIZATION

Amin Shali and Akanksha Jain
Problem Domain

- Multi-variable optimization problems
  - Thread Count
  - Blocking parameters
  - Pre-fetching Distance
  - Scheduling
  - Inner Loop Optimizations (Unrolling size)
  - TLB behavior
Problem Domain

- Unmanageable search space
  - Brute-force search
    - Expensive and intractable
  - Analytical Models
  - Machine Learning
    - Efficient methods to search intelligently
    - Automatic model construction
    - Less domain-specific knowledge
Machine Learning

- Supervised Learning
  - K-Nearest Neighbors
  - Artificial Neural Networks
  - Support Vector Machines
  - Kernel Machines
  - Statistical Machine Learning (Bayesian Learning)
  - Decision Trees
- Unsupervised Learning
- Reinforcement Learning
A Case Study

- Exploiting parallelism
  - Discovering parallelism
  - Expressing parallelism
  - **Mapping parallelism**

- Fixed heuristics – fail across architectures
- Analytical Models – low level hardware details
- Online learning models – Expensive learning

*Mapping Parallelism to Multi-cores: A Machine Learning Based Approach*
*Zheng Wang and Michael O'Boyle*
A Case Study

- The number of threads – Artificial Neural Network

- Selecting four Open MP scheduling policies – SVM
  - **BLOCK**: Iterations are divided into chunks of some fixed size (ceiling). Each thread is assigned a separate chunk.
  - **CYCLIC**: Iterations are divided into chunks of size 1 and each chunk is assigned to a thread in round-robin fashion.
  - **DYNAMIC**: Iterations are divided into chunks of some fixed size. Chunks are dynamically assigned to threads on a first-come, first-serve basis as threads become available.
  - **GUIDED**: Chunks are made progressively smaller until the default minimum chunk size (1) is reached.
Speedup on Xeon Platform for various scheduling policies give the no. of threads
Speedup on Cell for the cyclic scheduling policy for different number of threads
A Machine Learning Approach

- **Training**
  - Off-line training at the factory

- **Deployment**
  - Apply trained models for prediction
Feature Selection

- Features - inputs of a machine learning model
- Start with features that might be important
- Small feature sets are better
  - Learning algorithms run faster
  - Are less prone to over-fitting the training data
  - Useless features can confuse learning algorithms
Feature Selection

Static Code Features
- Operations
- Branches
- Memory operations

Dynamic features
- Loop iteration count
- L1 Dcache miss rate

Runtime feature
- Parallel execution time

Extracted from LLVM IR

Profiled with the smallest input data set
Artificial Neural Networks

- Computational model of biological neurons
- Network of simple processing elements
  - Complex behavior exhibited by their connections
- Capable of modeling both linear and non-linear functions
- Function approximation
- Classification
- Robust to noise
Function Approximation
Function Approximation
Over-fitting
A multi-layered Perceptron
Learning with an ANN

- **Feature Selection**
  - Set of variables representative of the output
  - Curse of dimensionality

- **Supervised learning**
  - Example inputs with correct output
  - Infer function by minimizing mean-squared error
  - Back-propagation (gradient descent)
Performance of the ANN based model
Support Vector Machines

- Supervised learning for classification
- Constructs hyper-planes in a higher dimensional space to achieve good separation
- Maximum-margin hyper-plane
- Kernels for non-linear classifiers

- Drawback: Multi-class classification done by reducing the task to several binary problems
- BLOCK scheduling
- CYCLIC scheduling
- DYNAMIC scheduling
- GUIDED scheduling

Hyperplane-3
Hyperplane-2
Hyperplane-1
Figure 7. The new program is profiled to extract features ($x$) which are fed into the ANN and SVM models respectively. The ANN model predicts the speedup ($s$) and the SVM model predicts the best scheduling policies ($p$) with different thread numbers.
Sensitivity to input

- Data insensitive predictor
  - Profile the program once with each input

- Data Sensitive Predictor
  - Profile the program once with the smallest input
  - Program behavior with limited variability across data sets at a fine-grain parallel level
Evaluation

- Stability and Portability
  - Performance across all programs
  - Performance on different architectures

- Profiling Overhead
  - Overhead in extracting features as compared to analytical models

Methodology — leave-one-out cross validation
Performance on Xeon

The performance ratio is averaged across data sets for each parallel loop.
# Profiling Cost

<table>
<thead>
<tr>
<th>Model</th>
<th>Profiling with the sequential program</th>
<th>Profiling with the parallel program</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DS</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>Regression-based</td>
<td>N</td>
<td>M * N</td>
</tr>
<tr>
<td>MMGP</td>
<td>N</td>
<td>M * N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Intel</th>
<th>Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>4.79sec</td>
<td>4.80mins</td>
</tr>
<tr>
<td>DS</td>
<td>13.10mins</td>
<td>1.75hours</td>
</tr>
<tr>
<td>Regression-based</td>
<td>59mins</td>
<td>16.45hours</td>
</tr>
<tr>
<td>MMGP</td>
<td>N.A.</td>
<td>41hours</td>
</tr>
</tbody>
</table>
Archana Ganapathhi, Kaushik Datta, Armando Fox and David Patterson
2009
Problem Description

- Different scientific parallel algorithm
- Many multi-core architectures
- Compilers alone
- Low performance
Solution: Auto Tuning

- What does it do:
  - Identifies optimization features
  - Search the parameter space for best performing configuration
- Cross platform
- Easily scalable
- Minimize programmer tuning time
Auto Tuning Cons

- Size of parameter space to explore is very big even in constrained circumstances
  - Solution: heuristics to prune the space
- Only tries to minimize running time
Statistical Machine Learning (SML)

- Draws inferences from automatically constructed models of data
- Methodologies include:
  - Bayesian Learning
  - Markov and Graphical Models
  - Regression
  - Kernel methods
- Allow to simultaneously tune for multiple metrics
Proposed Solution

- Using SML techniques to explore a larger parameter space faster
- Effectively identifying the relationships between optimization parameters and performance metrics
- Technique used: Kernel Canonical Correlation Analysis (KCCA)
Canonical Correlation Analysis (CCA)

- Find linear combinations of two sets of variables which have the maximum correlation with each other.
- Two sets of variables; X and Y.
- Objective:
  - Find A and B such that the correlation of AX and BY is maximized.
- Useful when the relationship of variables can be captured in linear form.
Maximize the correlation between U and V
Kernel Canonical Correlation Analysis (KCCA)

- Seeking the same objective as CCA
- Used to find non-linear relationship between the two sets of variables
- Useful when the variables have nonlinear relations
- Challenge: defining the kernel functions
Kernel Trick!

Non-linear relation

Kernel Trick

Linear relation
Maximize the correlation between U and V
Applying kernel functions to $X$ and $Y$.

Maximally correlated

Projected space
Overview of the Problem

Scientific Code

Configuration Parameters

Platform

Performance Parameters
### Configuration Parameters

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Parameter Type</th>
<th>Name</th>
<th>Parameter Range</th>
<th>Number of Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Thread Count</strong></td>
<td>Number of Threads</td>
<td>NTthreads</td>
<td>{2^0…2^3}</td>
<td>4</td>
</tr>
<tr>
<td><strong>Domain Decomposition</strong></td>
<td>Block Size</td>
<td>CX</td>
<td>{2^7…NX}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CY</td>
<td>{2^1…NY}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CZ</td>
<td>NZ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chunk Size</td>
<td></td>
<td>{1…NX \times NY \times NZ / 2^{CX \times CY \times CZ \times NTthreads}}</td>
<td></td>
</tr>
<tr>
<td><strong>Software Prefetching</strong></td>
<td>Prefetching Type</td>
<td></td>
<td>{register block, plane, pencil}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Prefetching Distance</td>
<td></td>
<td>{0, 2^5…2^9}</td>
<td>6</td>
</tr>
<tr>
<td><strong>Padding</strong></td>
<td>Padding Size</td>
<td></td>
<td>{0…31}</td>
<td>32</td>
</tr>
<tr>
<td><strong>Inner Loop Optimizations</strong></td>
<td>Register Block Size</td>
<td>RX</td>
<td>{2^0…2^1}</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RY</td>
<td>{2^0…2^1}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RZ</td>
<td>{2^0…2^3}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Statement Type</td>
<td></td>
<td>{complete, individual}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Read From Type</td>
<td></td>
<td>{array, variable}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Pointer Type</td>
<td></td>
<td>{fixed, moving}</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Neighbor Index Type</td>
<td></td>
<td>{register block, plane, pencil}</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>FMA-like Instructions</td>
<td></td>
<td>{yes, no}</td>
<td>2</td>
</tr>
</tbody>
</table>

The total number of configurations is greater than \(2^{25}\).
### Performance Metrics

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_TOT_CYC</td>
<td>Cycles per thread per job</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
<td>L1 data cache misses per thread</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>L2 data cache misses per thread</td>
</tr>
<tr>
<td>PAPI_TLB_DCM</td>
<td>TLB misses per thread</td>
</tr>
<tr>
<td>PAPI_CA_SHR</td>
<td>Accesses to shared cache lines</td>
</tr>
<tr>
<td>PAPI_CA_CLN</td>
<td>Accesses to clean cache lines</td>
</tr>
<tr>
<td>PAPI_CA_ITV</td>
<td>Cache interventions</td>
</tr>
<tr>
<td>Power meter</td>
<td>Watts consumed per second</td>
</tr>
</tbody>
</table>

\[
Y = \frac{\text{total cycles} \times \# \text{ of flops}}{\text{clk rate} \times \# \text{ of watts}}
\]

<table>
<thead>
<tr>
<th>Total Cycles</th>
<th>L1_DCM</th>
<th>L2_DCM</th>
<th>TLB_DCM</th>
<th>CA_SHR</th>
<th>CA_CLN</th>
<th>CA_ITV</th>
<th>Energy Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.9E7</td>
<td>2.4E5</td>
<td>1.5E5</td>
<td>1.2E4</td>
<td>1.2E5</td>
<td>1.4E4</td>
<td>1.2E3</td>
<td>2.3E4</td>
</tr>
</tbody>
</table>
Case Study: Stencil Code

- Sample 1500 datapoints from configuration space
- Run stencil with chosen configurations
- Identify relationship between optimization configurations and performance
- Manipulate relationship to find best performing configuration
Projecting Features

Configuration features

Performance Features

Raw Data Space

Project onto dimensions of maximum correlation

Projected Data Space

$K_x^*A$

Same Neighbors

$K_y^*B$
Kernel Functions

- Euclidean distance is not appropriate
- Custom measures
  - Numeric columns: Gaussian kernels
    - \( K(x_i, x_j) = \exp(-\|x_i - x_j\|^2/\tau) \)
    - \( \tau \) is calculated based on variance of the norms of data points
  - Non-numeric columns:
    - \( K(x_i, x_j) = 1 \) if \( x_i = x_j \), 0 otherwise
- Similarity of X1 and X2 = average(\( K(X1_i, X2_i) \))
... Projections

Optimization Configurations

Stencil Code

Platform

Performance

Kernel function

K_x

K_y

KCCA

\[
\begin{bmatrix}
0 & K_x K_y \\
K_y K_x & 0
\end{bmatrix}
\begin{bmatrix}
A \\
B
\end{bmatrix}
=\lambda
\begin{bmatrix}
K_x K_x & 0 \\
0 & K_y K_y
\end{bmatrix}
\begin{bmatrix}
A \\
B
\end{bmatrix}
\]

K_x^*A

K_y^*B
Finding Optimal Configuration

[Diagram showing configuration features and performance metrics in a raw data space, followed by an inverse image problem, then mapped to KCCA data space, and finally showing nearest neighbors in KCCA space.]

- Configuration features
- Performance Metrics
- Raw Data Space
- Inverse Image Problem
- KCCA Data Space
- Nearest Neighbors
- Kx*A
- Ky*B
... Finding Optimal Configuration
Genetic Algorithm

Initial Population

Fitness Evaluation

Selection

Reproduction

Modification
AMD Barcelona

Barcelona Performance

- No Optimization
- Expert Optimized
- Random Raw Data
- Genetic on Raw Data
- SML Optimized

GFlops/second

7 point stencil  27 point stencil
Intel Clovertown

Clovertown Performance

- No Optimization
- Expert Optimized
- Random Raw Data
- Genetic on Raw Data
- SML Optimized

GFlops/second

7 point stencil  27 point stencil
Intel Clovertown

Clovertown Energy Efficiency

- No Optimization
- Expert Optimized
- Random Raw Data
- Genetic on Raw Data
- SML Optimized

MFlops/sec/Watt

7 point stencil vs. 27 point stencil
Performance vs Energy Efficiency

Cloverfield Perf. vs. Energy Efficiency

- 8 threads
- 4 threads
- 2 threads
- 1 thread

GFlops/second vs. MFlops/sec/Watt
Conclusion

- Feature Extraction
- Generating training data is a time-consuming task and defines the prediction quality
- Tweaking of learning parameters
- Adaptive execution of programs for performance and energy
- Self-optimization
- Methods for a broader range of parallel applications