Partitioning Streaming Parallelism for Multicores: A Machine Learning Based Approach

Zheng Wang and Michael O'Boyle

PACT
Vienna, Austria, September, 2010
Background

- **Applications:**
  - Stream programs written by high level programming languages

- **The problem:**
  - Partitioning streaming applications for multi-cores

* We focus only on the partitioning stage in this work!
Why not Partitioning Heuristics?

- Hardwired heuristics might not adapt to different platforms

*Baseline is a naïve graph partitioner.

* The best partitioning strategy varies across programs and platforms.

(a) Intel Xeon 4-core (2x dual-cores)
(b) Intel Xeon 8-core (2x quad-cores)
Why Machine Learning?

- It can automatically learn from data; then, reuses the learned knowledge

Advantages:
- Doesn’t require expert knowledge and is portable across platforms

We use supervised learning technique in this work
- **Classical** supervised learning directly predicts the partitioning sequence
This is a Different Machine Learning Problem

- Previous machine learning based work limits on *fixed* targets
  - Determining the compiler flag settings
    - (Cavazos et al., CGO'07; Hoste and Eeckhout, CGO'08; Dubach et. al, MICRO'09)
  - Determining loop unroll factors
    - (Mark and Saman, CGO'05)
  - #Threads per parallel loop
    - (Wang and O'Boyle, PPoPP'09)

- We are dealing with a problem with *unbounded* graph structure

- The graph structure changes after each operation
The Challenge

- The partitioning sequence is **unbound** and the graph changes after every operation.

What happen if step-2 had not taken place?

* It is difficult for a predictive model to make sure each intermediate stage is 100% correct!
Our Two-Step Approach

Step 1. Predict features of the ideal partitioning structure

Step 2. Search and generate a legal partition as closed to the features of the predicted structure as possible.
Using Program Features to Characterise the Original Program and a Partition

<table>
<thead>
<tr>
<th>Static program features</th>
<th>features of the original program:</th>
<th>features of the ideal partition:</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Filters</td>
<td>#Joiners</td>
<td>[#filters; ...; work balance]</td>
</tr>
<tr>
<td>Pipeline depth</td>
<td>Splitjoin width</td>
<td></td>
</tr>
<tr>
<td>Avg. unit work</td>
<td>Max unit work</td>
<td></td>
</tr>
<tr>
<td>Pipeline work</td>
<td>Splitjoin work</td>
<td></td>
</tr>
<tr>
<td>Computation</td>
<td>Computation of stateful filters</td>
<td></td>
</tr>
<tr>
<td>Branches per inst</td>
<td>Load/Store per inst</td>
<td></td>
</tr>
<tr>
<td>Avg. communication rate</td>
<td>Computation-communication ratio</td>
<td></td>
</tr>
<tr>
<td>Avg. commun. / unit work</td>
<td>Avg. bytes commun. / unit work</td>
<td></td>
</tr>
<tr>
<td>Max commun. /unit work</td>
<td>Work balance</td>
<td></td>
</tr>
</tbody>
</table>

Machine Learning Model
Generate Training Data

- Using features to characterise both the program and generated partitions
  - Training is performed off-line using a set of training programs.

A training example:

Features of the original program
[feature1;...;featuren]

Features of the partition
[feature'1;...;feature'n]

Execution time

*The key point is to summarise features of the ideal partitioning structure.*
Summarise the Ideal Partitioning Structure (Training)

- Using K-Means clustering algorithm to summarise the best partition of a program

* The multi-dimensional feature space has been projected to a two-dimensional one.
Train and Use the Model

- Nearest-Neighbour Algorithm
  - Pick a program from the training set, whose features most closely match the input program’s features
Searching and Generating a Legal Partition

The predicted ideal partitioning structure (features)

The nearest generated partition

Randomly generate partitions and select the partition nearest the predicted ideal structure.
The Compilation Framework

Program Source → **Frontend** → **Machine Learning Partitioner** → **Backend** → **Runtime**

- **Extracted program features**
- **Predicted ideal partitioning structure features**

- **Machine Learning Model**

- **Partition generator**

- **Optimised binary**

**Execution**

- **Predicting the ideal structure**
- **Searching** (without executing any programs)
Experimental Setup

- **Platforms**
  - 2 x Dual Intel Xeon 5160 processors (4 cores in total)
  - 2 x Quad Intel Xeon 5450 processors (8 cores in total)

- **Comparison:**
  - 2 StreamIt compiler built-in partitioners
  - An analytical-based model (*Navarro et al.*, *PACT 2009*)

- **Compilers:**
  - StreamIt version 2.1.1
  - Intel ICC v11.0
    - -O3 -xT -aXT -ipo

- **Evaluation Methodology:**
  - Leave-one-out-cross-validation
    - Making sure the model has NOT seen the target program before.
  - Baseline: StreamIt dynamic programming-based partitioner

- **Benchmarks:**
  - 17 StreamIt applications
Results on the Intel 4-Core Platform (1.9x)

- Our approach achieves 1.9x speedup over the StreamIt default scheme.

*Baseline: StreamIt dynamic-programming-based partitioner*
Our approach achieves 60% of the Best-Found performance

*Baseline: StreamIt dynamic-programming-based partitioner
Similar Programs Have Similar Ideal Partitioning Structures

- Correlation of programs and the ideal partitioning structures

*The original multi-dimensional feature space have been projected into a single value.*
Adapt to the Intel 8-core Platform (1.8x)

- Our approach achieves 1.8x speedup over the StreamIt default scheme

![Diagram showing speedup comparison]

*Baseline: StreamIt dynamic-programming-based partitioner

- Analytic
- Greedy
- Machine Learning
Conclusions

- A machine learning based approach for partitioning streaming applications
  - The model is firstly trained off-line
  - Predicting the ideal partitioning structure used the trained model
  - Searching a legal partition closest to the predicted structure (without running any code)

- Comparison with heuristics and analytical-based approaches
  - Better performance and more stable across programs and platforms
  - An automatic and portable scheme
Thank You