A Large-Scale Cross-Architecture Evaluation of Thread-Coarsening

Alberto Magni, Christophe Dubach, Michael O'Boyle
Introduction

- Wide adoption of GPGPU for HPC
- Many GPU devices from many of vendors

AMD  Nvidia  Intel  Qualcomm  ARM
Introduction

OpenCL

AMD  Nvidia  Intel  Qualcomm  ARM
OpenCL is Functionally Portable
Performance Evaluation

OpenCL
---
Thread Coarsening

AMD     | Nvidia     | Intel

Regression Trees
---

- **Loads < 0.92**
  - T: 1.70
  - F: 0.40

- **Branches < 2.80**
  - T: 1.06
  - F: 0.81

**Cache Misses < 1.00**
Performance is NOT Portable

![Graph showing performance comparison between AMD Cypress and Nvidia Fermi](image-url)
What's next

- Motivation
- Compiler Infrastructure

**Thread-Coarsening**
- Experiments
- Data Analysis
- Conclusions and Future Work
Thread Coarsening

Original Thread Space

Transformed Thread Space

Reduce thread number

Increase amount of work
Advantages of Thread Coarsening

- Reduce the amount of redundant computation

- Perfect for a cross-architectural evaluation
  - Supported by the standard
  - Architecture independent
Our Portable Compiler

OpenCL

Clang

LLVM-IR

Transformations

Axtor

Transformed OpenCL

Proprietary Compiler

AMD

Nvidia

Intel
Thread Coarsening Implementation

- LLVM function pass replicates the instructions in the kernel body

\[
\begin{align*}
id &= 0 \\
\text{for index in (0 : width)} & \quad \text{for index in (0 : width)} \\
\quad \text{tmp} & \quad \text{tmp} \\
\quad \text{+= A[id + index];} & \quad \text{+= A[id + index];} \\
\quad \text{B[id] = tmp;} & \quad \text{B[id] = tmp;} \\
\end{align*}
\]
Thread Coarsening Implementation

Identify **divergent instructions**

\[
\begin{align*}
id &= 0 \\
\text{for index in (0 : width)} & \quad \text{for index in (0 : width)} \\
\quad \quad \text{tmp} &= \text{A[id + index]}; \\
\quad \quad \text{B[id]} &= \text{tmp}; \\
\text{id} &= 1 \\
\text{for index in (0 : width)} & \quad \text{for index in (0 : width)} \\
\quad \quad \text{tmp} &= \text{A[id + index]}; \\
\quad \quad \text{B[id]} &= \text{tmp};
\end{align*}
\]
Thread Coarsening Implementation

Replicate divergent instructions

\[
\text{id} = 0
\]

\[
\text{for index in (0 : width)}
\]
\[
\quad \text{tmp1} += A[\text{id} + \text{index}]
\]
\[
\quad \text{tmp2} += A[2*\text{id} + 1 + \text{index}]
\]
\[
\text{B[\text{id}]} = \text{tmp1}
\]
\[
\text{B[2*\text{id} + 1]} = \text{tmp2}
\]

Thread number is reduced at runtime
What's next

- Motivation
- Compiler Infrastructure
- Thread-Coarsening

- Experiments
  - Data Analysis
  - Conclusions and Future Work
Parameter Space

Static Parameters

- Coarsening
  - Factor
  - Stride
  - Direction

Dynamic Parameters

- Local Work Group Size

~300 configs for One-D benchmarks
~2,000 configs for Two-D benchmarks
Experimental Set-Up

- 17 benchmarks from Nvidia / AMD / Parboil
- 5 Devices:
  - Nvidia Fermi – GTX480
  - Nvidia Kepler – K20
  - AMD Cypress – HD 5900
  - AMD Tahiti – 7970
  - Intel Core-i7

\[ \sim 43,000 \text{ runs in Total} \]
Experimental Set-Up

• 17 benchmarks from Nvidia / AMD / Parboil

• 5 Devices:
  – **Nvidia Fermi** – GTX480
  – Nvidia Kepler – K20
  – **AMD Cypress** – HD 5900
  – AMD Tahiti – 7970
  – Intel Core-i7
Performance Varies Significantly

Nvidia Fermi | AMD Cypress

Speedup

binarySearch | blackholes | convolution | dwtHaar1D | fastWalsh | floydWarshall | mriQ | mt | mLocal | mvCoal | mvUncoal | nbody | reduce | sgenmm | sobel | spmv | stencil | geomean
Performance Varies Significantly

Nvidia Fermi
Performance Varies Significantly

AMD Cypress

![Graph showing speedup for various applications on AMD Cypress]
What's next

- Motivation
- Compiler Infrastructure
- Thread-Coarsening
- Experiments

- Data Analysis
- Conclusions and Future Work
Data Collection – Full Exploration

![Graph showing speedup vs. coarsening factor with points scattered across the graph. The axes are labeled Speedup on the y-axis and Coarsening Factor on the x-axis. There are lines indicating Stride and Local Size.]
Data Collection – Config Selection

![Graph showing speedup vs. coarsening factor](image-url)
Data Collection – Profiling

![Graph showing speedup vs. coarsening factor.](image-url)
Data Collection – Profiling

![Graphs showing speedup vs coarsening factor for different datasets](image-url)
Profiler Counters

Nvidia

- #instructions
  - #branches
  - #loads
  - #stores

- Cache:
  - L1 HitRate
  - L2 HitRate

AMD

- ALU Utilization
- Vector Utilization
- WLIW Packing
- Cache Utilization
- Memory Unit Utilization
Counters Analysis

**GOAL:** Discriminate fast and slow configs

**Speedup**

**Counter Relative Value**

- counter > x
- counter < x
Explaining Performance per Device

- Discriminate fast and slow configs
- Relate counters to performance
- Trees are easy to read
Tree Analysis

Nvidia Fermi

- Loads < 0.92
  - T: 1.70
  - F: 1.06

- Branches < 2.80
  - T
  - F: 0.40

- Cache Misses < 1.00
  - T
  - F: 0.81
Tree Analysis

Nvidia Fermi

Speedup

![Speedup Chart](chart.png)

- 1.70 for floydWarshall
- 1.70 for sgemm

Loads < 0.92

T

F

...
Dynamic Counter

Number of Loads

Coarsening Factor

floydWarshall

sgemm
Tree Analysis

Nvidia Fermi

Tree Diagram:

- **Loads < 0.92**
  - T
  - F

  **1.70**
  - floydWarshall
  - sgemm

- **Branches < 2.80**
  - T
  - ... (Remaining branches)
  - F

  **0.40**
  - spmv
  - mvCoal
Dynamic Counter

Number of Branches

mvCoal

spmv

Coarsening Factor
Trees Analysis

AMD Cypress

ALUPacking < 1.28

T

0.8

T

ALUBusy < 0.59

F

2.10

0.79

spmv

stencil

nbody

BinarySearch

mt
Dynamic Counter

![Graph showing the relationship between ALUPacking and Coarsening Factor for different tasks (nbody, binarySearch, mtLocal).](image)

- **ALUPacking** on the y-axis.
- **Coarsening Factor** on the x-axis ranging from 1 to 32.
- Three lines representing different tasks:
  - **nbody**
  - **binarySearch**
  - **mtLocal**

The graph illustrates how the performance (ALUPacking) varies with the coarsening factor for each task.
What's next

- Motivation
- Compiler Infrastructure
- Thread-Coarsening
- Experiments
- Data Analysis

- Conclusions and Future Work
Conclusion and Future Work

- Automatic methodology for performance explanation
- First step toward definition of compiler heuristics and automatic coarsening tuning