Compiler Optimisation for GPGPUs

Michael O’Boyle
University of Edinburgh
Outline of talk

- Who am I and what do I do
- Compilers
  - what and why they disappoint
  - a data driven approach to their improvement
- OpenCL optimisation for GPUs
  - thread coarsening on 4 platforms
Who am I

- University of Edinburgh
  - ICSA, CArD, CDT
  - 1 year sabbatical
- Research
  - Mapping parallelism
  - Heterogeneous esp. GPGPU
  - DSE of architecture
- ARM
  - Centre of Excellence at Edinburgh.
  - HiPEAC
  - Trondheim: Learn about GPU hardware. Help improve compilers
View of compilers

Floors 2–5

Floor 1

Applications

Programming Language

Compilation

Runtime System

Computer Architecture

Physical Realisation

Wednesday, 8 October 2014
View of compilers
Optimization for scalar machines is a problem that was solved ten years ago [Kuck 1990]

Parallel compilation is a dead end
- it will be solved in 5 years [My supervisor 1990]

I don’t want the compiler to do automatic data distribution - that’s the fun part [Chemist UMan 1992]
View of compilers

Optimization for scalar machines is a problem that was solved ten years ago [Kuck 1990]

Parallel compilation is a dead end - it will be solved in 5 years [My supervisor 1990]

I don’t want the compiler to do automatic data distribution - that’s the fun part [Chemist UMan 1992]

CONF is for smart hardware and dumb compilers there’s no place for smart compilers here [Review 2005]
Optimization for scalar machines is a problem that was solved ten years ago [Kuck 1990]

Parallel compilation is a dead end - it will be solved in 5 years [My supervisor 1990]

CONF is for smart hardware and dumb compilers there’s no place for smart compilers here [Review 2005]

I don’t want the compiler to do automatic data distribution - that’s the fun part [Chemist UMan 1992]

One of the most technical domains for ml [ML academic 2004]

Perfect research topic: always room for improvement as it is undecidable [Anon 2008]
“Even if we assume that the beginning of useful compiler optimization research began in the mid 1960’s, the uniform performance improvement on integer intensive codes due to compiler optimization is still only **3.6% per year**.

This lies in stark contrast to the **60% per year** performance improvements we can expect from hardware due to Moore’s Law.”
Why do compilers fail?

- Wanted to see if we could improve performance
  - Huge gap between compiler and library/programmer

- We looked at matrix multiplication
  - world’s most studied compiler problem
  - 2 transformations
  - see how good existing approaches were

- What does the space look like?
  - Can knowledge help build better compilers
UltraSparc N=512

3D graph showing the relationship between unroll and tile size, with time as the vertical axis.

Wednesday, 8 October 2014
UltraSparc: within 20% of minimum N=400

Minimum at: Unroll = 3 and Tile size = 57. Near minimum: 2.6%. Original 4.99 secs, Minimum 0.56 secs
R10000 within 20% of the minimum N=512

Minimum at: Unroll = 4 and Tile size = 85. Near minimum: 7.2%. Original 2.79, Minimum 1.09

Wednesday, 8 October 2014
Minimum at: Unroll = 17 and Tile size = 51. Near minimum: 4.6%.
Original 30.93, Minimum 3.34
Main conclusions

- Space is hard
  - especially if hardware changes
- All Existing Compiler Analysis
  - FAILED
  - most studied benchmark
- Need empirical evidence
  - rather than theory
- A massive cultural shift
- 2 ways forward
  - Search
  - Machine Learning
Just Search

• Known as iterative compilation
  – or auto-tuning in library space

• Platform blind
  – No hard wired heuristics
  – Based on evidence!
  – not belief

• However
  – expensive
  – redo when anything changes
Why not use knowledge of other programs?

New Program

Simplest form of machine learning
Prior knowledge speeds up search
Compiler writer model
Compiler writer model

Fiction

Reality
Given enough data and right model

- Can automatically capture behaviour
- Can predict outcomes
- Without knowing internal
Lessons learnt

- Hand crafted optimisation
  - doomed to fail
  - based on fiction

- Let go of certainty
  - dynamic analysis
  - statistical modelling
  - automatic prediction via ml

- Hardware is probabilistic
  - why not software?
Portable OpenCL compiler

- OpenCL popular language
  - Portable but not performance portable
- Opaqueness of GPUs makes predicting performance difficult
  - Barrier to portable optimisation
- Want to build a portable optimiser
  - Our approach:
    - Source to source transformations
    - Auto-modelling
- Much harder problem than expected
  - Analysis needed
3 transformations
- Thread coarsening: using divergence analysis
- Stride optimisations
- Work group size

Can we use data driven approach to EXPLAIN behaviour? [SC13] and PREDICT optimisation [PACT14]
What is thread Coarsening?

```c
kernel void matrixTransposition(
    int width, height, global float* in, out)
{
    uint column = get\_global\_id(0);
    uint row = get\_global\_id(1);

    uint inputIdx = row * width + column;
    uint outputIdx = height * column + row;

    out[outputIdx] = in[inputIdx];
}
```
What is thread Coarsening?

```c
kernel void matrixTranspositionCoarsened2x(
    int width, height, global float* in, out)
{
    uint column = get_global_id(0);
    uint row0 = 2 * get_global_id(1): + 0:;
    uint row1 = 2 * get_global_id(1): + 1:;

    uint inputIdx0 = row0 * width + column;
    uint inputIdx1 = row1 * width + column;
    uint outputIdx0 = height * column + row0;
    uint outputIdx1 = height * column + row1;

    out[outputIdx0] = in[inputIdx0];
    out[outputIdx1] = in[inputIdx1];
}
```
Coarsening helps: Matrix Transpose

**Speedup**

- Cypress
- Tahiti
- Core–i7
- Fermi
- Kepler

**Coarsening Factor**

1 2 4 8 16 32
43,000 experiments

- 17 Benchmarks: Parboil and SDKs
- 5 Platforms
  - Nvidia: Fermi, Kepler
  - AMD: Radeon and Cypres
  - Intel CPU: Core i7

Parameter |
| Possible values |
| Coarsening factor |
| \{1, 2, 4, 8, 16, 32\} |
| Coarsening dimension |
| \{0, 1\} |
| Stride |
| \{1, 2, 4, 8, 16, 32\} |
| Local work group size |
| \{\textit{dev min}, \ldots, 32, 64, 128, \ldots, \textit{dev max}\} |
Large Potential

Comparison against hand-coded

Large potential

0.5
1
1.5
2

Core-i7
Fermi
Kepler
Cypress
Tahiti

Speedup

Original local size
Best local size
Coarsening only
Coarsening + stride

Wednesday, 8 October 2014
Comparison against hand-coded C

Speedup Distribution

Distributions look fine at coarse level

Fermi
Kepler
Core-i7
Tahiti
Cypress

Wednesday, 8 October 2014
Hard to find: Fermi

Speedup

Fermi

binSearch
blackholes
correlation
dwtHaar1D
fastWalsh
floydWarshall
mriQ
mt
mtLocal
mvCoal
uncoal
bodies
reduce
gemm
sobel
spmv
stencil

Wednesday, 8 October 2014
Harder to find: Kepler

Speedup

Kepler

binarySearch blacksholes convolution dwtHaar1D fastWalsh floydWarshall miQ mt mtLocal myCoal mvUncoal nbody reduce sgemm sobel spmv stencil
Hard to find: Cypress

Comparison against hand-coded Cypress

- Binary Search
- Convolution
- DWT Haar 1D
- Fast Walsh
- Floyd-Warshall
- MRI Q
- MT
- MT Local
- MV Coal
- MV Uncoal
- N-body
- Reduce
- Sgemm
- Sobel
- Spmv
- Stencil

Speedup

Wednesday, 8 October 2014
Hard to find: Tahiti

<table>
<thead>
<tr>
<th>Library</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarySearch</td>
<td>4.59</td>
</tr>
<tr>
<td>blacksholes</td>
<td>2.80</td>
</tr>
<tr>
<td>dwtHaar1D</td>
<td>3.05</td>
</tr>
<tr>
<td>fastWalsh</td>
<td></td>
</tr>
<tr>
<td>floydWarshall</td>
<td></td>
</tr>
<tr>
<td>mriQ</td>
<td></td>
</tr>
<tr>
<td>mt</td>
<td></td>
</tr>
<tr>
<td>mtLocal</td>
<td></td>
</tr>
<tr>
<td>mvCoal</td>
<td></td>
</tr>
<tr>
<td>mvUncoal</td>
<td></td>
</tr>
<tr>
<td>nbody</td>
<td></td>
</tr>
<tr>
<td>reduce</td>
<td></td>
</tr>
<tr>
<td>sgemm</td>
<td></td>
</tr>
<tr>
<td>sobel</td>
<td></td>
</tr>
<tr>
<td>spmv</td>
<td></td>
</tr>
<tr>
<td>stencil</td>
<td></td>
</tr>
</tbody>
</table>

Wednesday, 8 October 2014
Hard to find: i7

Hard to find: i7

Core–i7

Speedup

binarySearch
blackholes
convolution
dwtHaar1D
fastWalsh
floydWarshall
miQ
mt
mtLocal
myCoal
myUncoal
nbody
reduce
sgemm
sobel
spmv
stencil

Wednesday, 8 October 2014
Best Factors: Kepler

Kepler

Stride

1

2

4

8

16

32

binarySearch
blackholes
convolution
dlWHaar1D
fastWalsh
floydWalsh
mrI
mt
mtLocal
mvCoal
mvUncoal
nbody
reduce
sgemm
sobel
spmv
stencil

Wednesday, 8 October 2014
Best Factor: Cypress
Best Factor: Cypress

- Try to correlate performance to hardware counters
- Relate speedup to change in counters
- Aim
  - To provide insight to drive optimisations
- Used regression trees
  - Higher features provide most discrimination
Analysis: Fermi

loads < 0.92

Y

1.70

N

branches < 2.8

CacheMisses < 1.00

1.06

0.81

0.4

Wednesday, 8 October 2014
Kepler: Similar, less tolerant of branches

Diagram:

- loads < 0.92
  - Y: 1.69
  - N: branches < 1.06
    - CacheMisses < 1.00
      - 0.99
      - 0.76
    - 0.34
Analysis: Cypress

ALUPacking < 1.28

Y

N

0.8

ALUBusy < 0.59

0.79

2.10
Tahiti: Very different from Cypress

Tahiti:

Very different from Cypress

Analysis:

Wednesday, 8 October 2014
Analysis: i7  Loads unimportant

```
vectorInsts < 1.42

Y

branches < 0.7

N

ipc < 1

3.8

1.13

0.97

1.47
```
Predict based on analysis

- Compiler writers
  - Analysis fine but
  - Want to improve not just understand!
- Build a model using usual ml methodology to
  - Determine should I coarsen or not?
  - A factor 2 at a time
- Need to compare against default model
  - Average Best of N-1 model
Machine Learning in one slide

Collecting training data

Training programs → Exhaustive experiments → Best Configuration
Machine Learning in one slide

Collecting training data

- Training programs
- Exhaustive experiments
  - Best Configuration

Program features
- Machine Learning algorithm

Training the model

Wednesday, 8 October 2014
Machine Learning in one slide

Collecting training data
- Training programs
- Exhaustive experiments
- Best Configuration
- Program features
- Machine Learning algorithm

Prediction
- Predicted configuration
- Machine Learning model
- Program features

Training the model

Wednesday, 8 October 2014
60% of max performance
Summary

- OpenCl: compiler performance available
  - Complex space, tricky to predict
- Develop a “one-shot” predictor
  - Static and dynamic features
- More aggressive coarsening
  - Impact on larger training set
- Trans-architecture learning
  - Larger optimisation space
Thanks

Michael O’Boyle
University of Edinburgh