Towards Collaborative Performance Tuning of Algorithmic Skeletons
Uncontentious statement:
High level programming is great!
So why do application programmers resort to writing *low level* code?
So why do application programmers resort to writing low level code?

Performance!
common perception is ...
common perception is ...
programmers who care about performance

programmers who care about abstractions
programmers who care about *performance*

programmers who care about *abstractions*

GPGPU devs
GPGPU developers care about performance, whereas we care about abstractions.
How do we break the illusion?
High level code needs to be at least competitive with low level.
High level code needs to be at least competitive with low level (but faster would be nice)
Reasons for low level:
Reasons for low level:
Domain-specific optimisations
Reasons for low level:
Domain-specific optimisations
Parameter tuning
Reasons for low level: Domain-specific optimisations

Parameter tuning
Parameter tuning for Algorithmic Skeletons
Parameter tuning for Algorithmic Stencils
OpenCL workgroup size

Parameter tuning for Algorithmic Skeletons

Stencil Skeletons
OpenCL workgroup size:
OpenCL workgroup size: Controls decomposition of threads.
OpenCL workgroup size:

Controls decomposition of threads.

Is a 2D parameter (rows x cols).
OpenCL workgroup size:
Controls decomposition of threads.
Is a 2D parameter \((\text{rows} \times \text{cols})\). Critical to performance.
OpenCL workgroup size:

- rows
- cols
- performance
Examples
Same stencil!
Different device!
Same device!
Different stencil!
Choosing workgroup size depends on:

1. device
2. program
3. dataset
Let’s automate this!
Approach 1
Set a workgroup size
Execute and time program
Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
... (continue until done / bored)
Pick the best one you tried
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
... (continue until done / bored)
Pick the best one you tried
BAD!
Takes a loooong time
BAD!
Takes a loooong time

BAD!

Must be repeated for every new “X”

device
program
dataset
Approach 2
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
... (continue until done / bored)
Pick the best one you tried
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
... (continue until done / bored)
Pick the best one you tried
Collect **data points**
Extract “features”
Train machine learning classifier

Extract “features”
Input to classifier
Can make *predictions* on unseen “x”
Can make predictions on unseen “X”

BETTER!

Still takes a loooong time
Can make *predictions* on unseen “x”

Still takes a looooonng time

Requires a lot of code
Our wish list:

1. Reduce training costs
2. Reduce implementation costs
3. Minimise runtime overheads
Our Approach ...
OmniTune
1. Allows *collaborative* performance tuning

Reduce training costs ✓
2. Provides re-usable implementations

Reduce implementation costs ✓
3. Provides lightweight runtime interface

Minimise runtime overheads ✓
How does it work?
Remote

Servers

Clients
Remote

Book-keeper

Manages and stores training data
Remote

Servers

Clients
Servers

Autotuning engine

Performs machine learning
Remote

Servers

Clients
Target applications

Programs we want to tune

Clients
Remote

Servers

Clients
Features, param, performance
New training data
New training data
Remote

Servers

Clients
Demonstration
Implementation:

Remote: AWS instance + MySQL

Server: standalone system daemon, decision tree classifier

Client: modified SkelICL stencil pattern
Remote

TCP/IP

Servers

DBUS

SkelCL

...
Experimental Setup:
6 stencil benchmarks + synthetic.
7 different GPUs & CPUs.
4 dataset sizes.

Exhaustive search of workgroup size
space for each
Results
Performance

Intel i5-4570
Intel i5-2430M
Intel i7-3820
Nvidia GTX 690
AMD Tahiti 7970
Nvidia GTX TITAN
Nvidia GTX 590

$\Delta 0$
$\Delta 2$
$\Delta 4$
$\Delta 6$
$\Delta 8$

15.14x speedup best over worst
32% cases has unique optimal workgroup size
Classification performance

Speedup over human expert

10-fold  Synthetic  n-1 Device  n-1 Kernel  n-1 Dataset  Average
Classification performance

<table>
<thead>
<tr>
<th>10-fold</th>
<th>Synthetic</th>
<th>n-1 Device</th>
<th>n-1 Ke</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
</tbody>
</table>

Average 90% optimal performance

Speedup over human expert
Conclusions
High level GPU code must compete with low level on *performance*. That means *automating* the kind of tuning which is typical of low level.
We present a framework for doing this using machine learning.

Demonstrated using SkelCL stencils.

Achieves average 1.22x speedup over human expert.
Towards Collaborative Performance
Tuning of Algorithmic Skeletons

Cristian Calauze  Paucho Perastrello  Nicola Strozer
University of Edinburgh

Abstract
The physical instantiation of high-performance design have become an ever-growing problem, with the development of heterogeneous systems. In this paper, we describe efforts to attain performance through algorithmic skeletons, by leveraging the shared memory of modern CPU architectures. We also describe the development of an open-source framework to support the development of such applications. The framework is designed to provide a high-level interface for the development of such applications, allowing developers to focus on the algorithmic aspects of the problem, while leaving the low-level details to the framework.

1. Introduction
General-purpose computing with GPUs has been shown to provide significant speedups for many applications. We focus on algorithmic skeletons, which provide a high-level interface for the development of such applications, allowing developers to focus on the algorithmic aspects of the problem, while leaving the low-level details to the framework.

2. The Performance Problem
The primary challenge in the problem of accelerating applications on general-purpose GPUs is the communication overhead. This is a significant bottleneck, as the performance of the algorithmic skeletons is limited by the communication overhead.

3. Conclusion
In conclusion, we have demonstrated the effectiveness of our framework in accelerating applications on general-purpose GPUs. Our results show that the framework can provide significant speedups for a wide range of applications, making it a valuable tool for developers.

Details in the paper!
Towards Collaborative Performance Tuning of Algorithmic Skeletons

http://chriscummins.cc