Cost Models for Combinator Parallelism in GPGPUs

Stuart Monro
monros@dcs.gla.ac.uk

Collaboration Benefits

COST has funded collaboration between the University of Glasgow and the Chemnitz University of Technology

This has allowed us to combine the following:

Chemnitz

- Expertise in cost modelling and implementation
- Expertise in the development of programming models

Glasgow

- Expertise in functional programming & development of combinator algorithms

Both

- Expertise in parallel programming

Project that will be discussed in this talk requires deep knowledge in all of these areas to ensure success, collaboration is aiding in this

Motivation

High Level Techniques for Parallel Programming

with a focus on

Programming models and cost models for data parallelism

Aim is to produce a programming model offering the following benefits:

- Abstracting away from the architectural complexity of the GPU
- Supporting parallel programming on heterogeneous architecture
- A cost model to support program organisation decisions
- Integration with existing task parallel models (such as TwoL)

Background

- CPUs have hit a wall in speed improvements
- The key to improving performance now is parallelism
- Restructuring a sequential algorithm in order to parallelise it is not straightforward.

There is a large design space

- Multicores – task & data parallelism
- FPGAs – task & data parallelism
- CPU clusters – task & data
- GPUs – data (& task?)

Regardless of which approach we take we need multiple processors
GPUs are cost effective:

Tesla C Series 2075	€2,100
GTX 670M	€760
GTX 680	€500

Background

- Too expensive in terms of time & effort to code up different architectures to compare them, need a cost model to provide a comparison on which to base a decision

- This comparison can be provided by using a cost model to gain a rough understanding on likely performance improvement

- The accuracy required for the predicted performance improvement varies from case to case:
  - E.g. sequential program takes 9 months to run, when predicting performance improvement, accuracy of +/- 1 week is acceptable

The cost model can also help in terms of considering development time e.g.:
- CPU cluster will result in 15% speed up & will require 100 hours of development time
- GPU will result in 25% speed up & will require 500 hours of development time

If a 15% performance gain is sufficient for the programmer's needs then CPU cluster may be best solution

The cost model informs the decision

Cost Models

- A cost model expresses the execution time, memory consumption or power consumption of an algorithm as a function of relevant parameters

- It can be used to predict the performance of an algorithm based on those parameters

- For example consider the following:

```
int sourceArray[i];
int resArray[i];
int j = 42;

for (k = 0; k < i; k++)
        if sourceArray[k] < j
                resArray[k] = 1;
```

- The performance of a parallel algorithm to carry out this task could potentially depend on the following parameters:
  - Number of comparisons
  - Number of threads
  - Number and type of memory accesses
  - Cache behaviour

Cost Models

- Cost models can assist both in:
  - Choosing the architecture
  - Structuring the algorithm.
- They can be used by the programmer to make key decisions based on empirical data.
- There is a trade off between the usability and accuracy of the cost model:
  - The more accurate the model, the more complex it will be to use

A cost model does not need to be 100% accurate to be useful

Research Approach

- Select a complex system
- Take actual measurements of that system's performance
- Use curve fitting to develop a cost model

Why?

- The GPU is a complex system
  - Complicated to work out theoretical models based on architectural details
  - Architectural details are proprietary & not all are public
- GPU characteristics vary
  - Aim to develop portable experiments
  - Quickly applied to new chips when they become available
- Performance may be dependant on problem size
  - Need to know the constants in a model
  - These can only be identified through measurement

Methodology

The approach used to develop cost models

Different Architectures
(All Nvidia)

- GeForce GTX 590 - 512 CUDA cores – clock speed 1.22 GHz
- GeForce GT 520 - 48 CUDA cores – clock speed 1.62 GHz
- GeForce 8800 GTX - 128 CUDA cores – clock speed 1.35 GHz
- GeForce 8200 – 8 CUDA cores – clock speed 1.5 GHz

Different architectures used to identify whether models are portable or must be architecture-specific

Depending on the outcome, models will be developed accordingly

Methodology

The approach used to develop cost models

Measurements

GPU kernel (example – Logical AND operation)

```
dev_shared[threadIdx.x] = dev_dataArray[tid];

startComparison = clock();

result = dev_shared[threadIdx.x] && comparisonVal;

endComparison = clock();
elapsedTime = endComparison - startComparison;
dev_timingArray[tid] = elapsedTime;

if (result)
        dev_shared[threadIdx.x] = trueResult;
else
        dev_shared[threadIdx.x] = falseResult;

dev_dataArray[tid] = dev_shared[threadIdx.x];
```

Methodology

The approach used to develop cost models

Measurements

Timing done at thread level, focusing on the operation only

```
dev_shared[threadIdx.x] = dev_dataArray[tid];

startComparison = clock();

result = dev_shared[threadIdx.x] && comparisonVal;

endComparison = clock();
elapsedTime = endComparison - startComparison;
dev_timingArray[tid] = elapsedTime;

if (result)
        dev_shared[threadIdx.x] = trueResult;
else
        dev_shared[threadIdx.x] = falseResult;

dev_dataArray[tid] = dev_shared[threadIdx.x];
```

Methodology

The approach used to develop cost models

Measurements

Results of operation are recorded & later returned to the user to avoid compiler optimisations

```
dev_shared[threadIdx.x] = dev_dataArray[tid];

startComparison = clock();

result = dev_shared[threadIdx.x] && comparisonVal;

endComparison = clock();
elapsedTime = endComparison - startComparison;
dev_timingArray[tid] = elapsedTime;

if (result)
        dev_shared[threadIdx.x] = trueResult;
else
        dev_shared[threadIdx.x] = falseResult;

dev_dataArray[tid] = dev_shared[threadIdx.x];
```

Methodology

The approach used to develop cost models

Measurements

Repeated multiple times
For different sizes of source data set

Produces high number (between 100 – 1000) of result data sets to base cost models on

Curve Fitting

Matlab used to analyse all result data sets and produce basic models based on those. These models typically take the form:

$$f(x) = Ax + B$$

Where:

A & B are specified by Matlab
x is the number of operations to be performed

Methodology

The approach used to develop cost models

Model Checking

- "Clean" data sets produced
- Models checked against those for accuracy
- Approach for checking:
  - Automated script takes model as input values
  - Iterates through every entry in clean data set
  - Calculates predicted time for each entry using model
  - Compares prediction with actual

Accepted Margin Of Error

The output from model checking is the predicted time (PT) and the actual time (AT)

If $$|AT - PT| < \frac{AT}{10}$$

then it is considered that the model falls within the accepted margin of error

Methodology

The approach used to develop cost models

Model Finalisation

- One basic model developed for each data set
- Minor discrepancies normally seen in values of A & B
- Average out discrepancies
- Test finalised model against clean data sets

Models Produced to Date / Under Development

Data types models developed for:

- Integers – 4 bytes
- Doubles (where supported) – 8 bytes

Operations

- Add
- Multiply
- Compare
- Logical AND
- Logical OR

Data Transfer

- Host to device global memory
- Device global memory to host
- Device global to shared memory
- Device shared to global memory

Combinator Algorithms

- Capture methods & patterns of parallelism
- Roughly analogous to Java design patterns but with a mathematical foundation
- Can be difficult to implement, particularly in parallel architectures
- However many parallel algorithms can be expressed using combinators
  - This makes them a powerful programming methodology
  - Rewards the effort involved in implementation
- Aim is to implement some algorithms which use combinators and use cost models to predict their performance

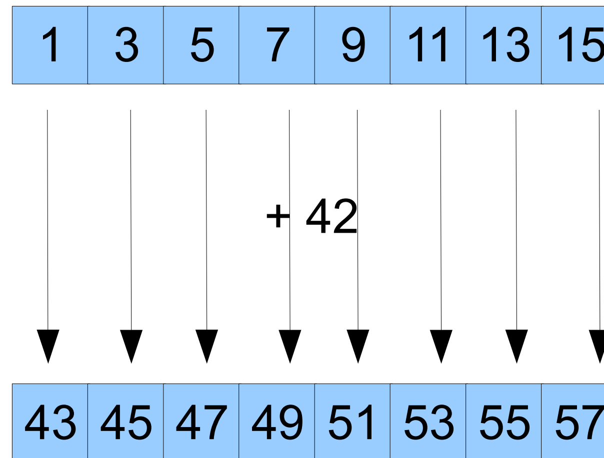
Combinator Algorithms

- Fundamental data structure – the array

Standard operations across an array:

Map

Iterate across array, adding 42 to each index position




```
for (i = 0; i < arrayLength; i++)
      array[i] += 42;
```

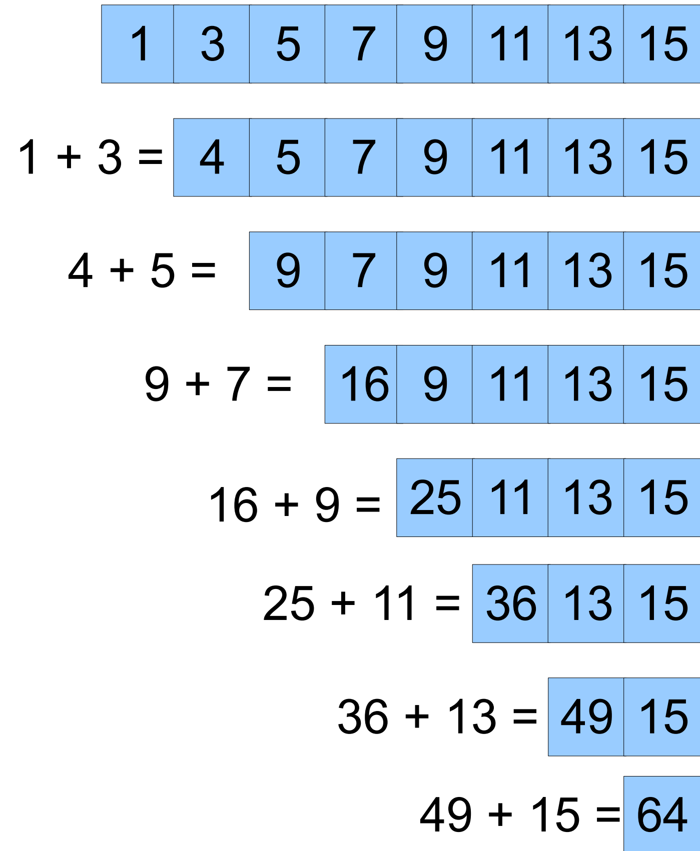
Combinator Algorithms

Standard operations across an array:

Fold

Iterate across array, calculate sum of all values

```
for (i = 0; i < arrayLength; i++)
      result += array[i];
```



Combinator Algorithms

Further array operations:

- Scan family
- Sweep

Which can be seen as a combination of maps & folds.

E.g. ScanL (scan from left or prefix sum):

Uses fold approach as illustrated in previous slide
Output is not a single value:

sourceArray

1	3	5	7	9	11	13	15

resultArray

1	4	9	16	25	36	49	64

```
for (i = 1; i < sourceArrayLength; i++)
    resultArray[i] = sourceArray[i-1] + resultArray[i-1];
```

Combinator Algorithms

Using these standard algorithms as building blocks we can construct many more algorithms and functions.

First we need to develop cost models for the basic functions

Key decision:

Which aspects of each function contributes to the cost model?

- Host code?
- Data transfer?
- Kernel call?
- Algorithm?

Case by case approach

Combinator Algorithms

Intended Cost Model for Map

Kernel code:

Fetch array from global memory into shared

Start timing

Add constant to each value in shared memory array

Stop timing

Write start & stop times to timing array

Write shared memory array back to global

Only the map function itself is timed as cost models for all other operations exist or are in development

Combinator Algorithms

Intended Cost Model for Fold (AKA Reduce)

More complex than map:

- Initial reduction is performed in shared memory on the device until original source array has been reduced to one value per block
- Those remaining values are copied back to the host
- Final reduction is performed on the host until a single value remains

The overall time required to perform the operation will consist of:

Time to perform reduction in shared memory on device
Time to perform final reduction on host

Time required to transfer data will not be recorded as cost models are being developed separately.

Future Work

- Model for Scan
- High level programming models
  - Level of abstraction
  - Libraries
  - Cost-model based decisions
- Ongoing collaboration to develop programming models