Project Two

Parallel computation of statistical values

Blair Cox, Logan Small and Michael Kemp

Group 12 - ParaStats

Introduction

Problem Context

"When working with data sets created from natural processes, it is often important to get statistical values for these datasets. This assignment requires you to implement the calculation of statistical values over large data sets in parallel, for example the average, median, standard deviation etc."

- Natural processes - A data stream of continuous values that are measuring events in nature.

Problem Context

"When working with data sets created from natural processes, it is often important to get statistical values for these datasets. This assignment requires you to implement the calculation of statistical values over large data sets in parallel, for example the average, median, standard deviation etc."

- Statistical values - Descriptive statistics that summarise a data set through representative scalar values.

Problem Context

"When working with data sets created from natural processes, it is often important to get statistical values for these datasets. This assignment requires you to implement the calculation of statistical values over large data sets in parallel, for example the average, median, standard deviation etc."

- Large data sets - Large amounts of data that would be inefficient to process through normal means.

Approach

- Parallelism
- OpenCL
- On-line processing

Analysing subsets of the data and then using reduction to combine the results.

Approach

- Parallelism
- OpenCL
- On-line processing

Low level programming language which generates portable code for heterogeneous devices.

Approach

- Parallelism
- OpenCL
- On-line processing

Dividing the input file into processable chunks and streamlining them onto the devices.

OpenCL

Motivation for OpenCL

Why does OpenCL exist?

- A modern system can contain multiple CPUs and GPUs
- Different vendors created different standards for utilizing their devices.
- It is difficult to utilize this heterogeneous platform for parallel computing
- OpenCL lets you create a single portable program that runs on all devices in parallel

What is OpenCL

What is OpenCL?

- Standardised language that doesn't tie implementation to specific devices.
- It is a programming framework for parallel computing.
- Runs instructions on acceleration units irregardless of hardware specifications.

Platform Model

- Made up of a host, and multiple OpenCL devices
- These devices are comprised of multiple compute units, and these compute units consist of processing elements.

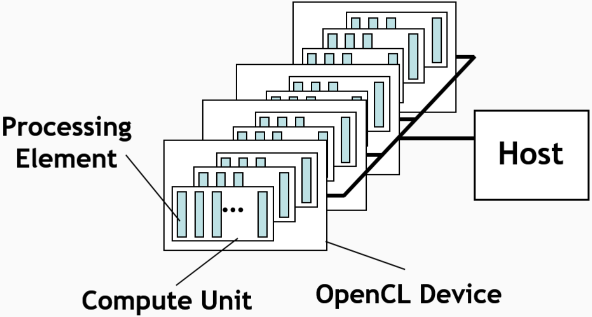

Figure 1: Platform Model ©*Khronos Group, 2013*

Data parallelism

Translate loops to functions (kernels) which operate on every data point in the problem domain

C Loop

```
void add(
  const int n,
  const float* a,
  const float* b,
  float* c)
{
  for (int i = 0; i < n; i++)
      c[i] = a[i] + b[i];
}
```

Data parallelism

Translate loops to functions (kernels) which operate on every data point in the problem domain

C Loop

```
void add(
  const int n,
  const float* a,
  const float* b,
  float* c)
{
  for (int i = 0; i < n; i++)
      c[i] = a[i] + b[i];
}
```

OpenCL Kernel

```
__kernel void add(
    __global float* a,
    __global float* b,
    __global float* c)
{
    int i = get_global_id(0);
    c[i] = a[i] + b[i];
}
```

N-Dimensional Work Domain

- When we structure the problem domain, we specify up to 3 dimensions and their sizes
- Each point in the specified domain is mapped to a work-item
- Work-items are grouped into work-groups which share local memory

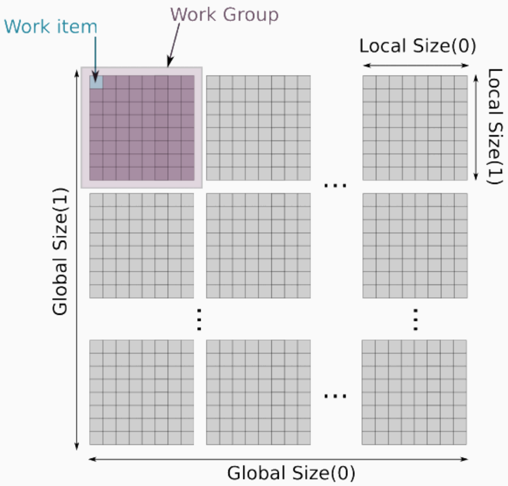

Figure 2: Work Item/Group Layout [Hong et al., 2016]

Memory Model

Work-item	$\longrightarrow$	Private Memory
Work-group	$\longrightarrow$	Local Memory
Compute Device	$\longrightarrow$	Global Memory
Host Device	$\longrightarrow$	Host Memory

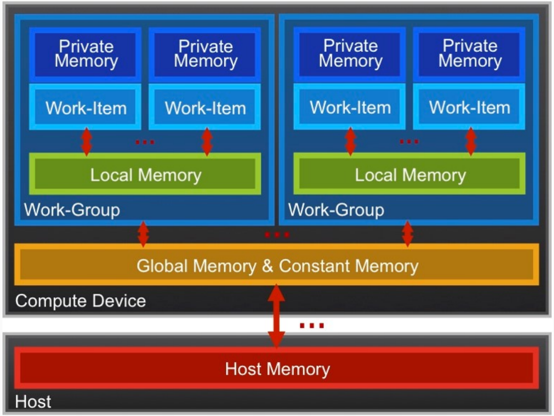

Figure 3: OpenCL Memory Model ©*Khronos Group, 2013*

GPU vs CPU

- Matrix multiplication performance

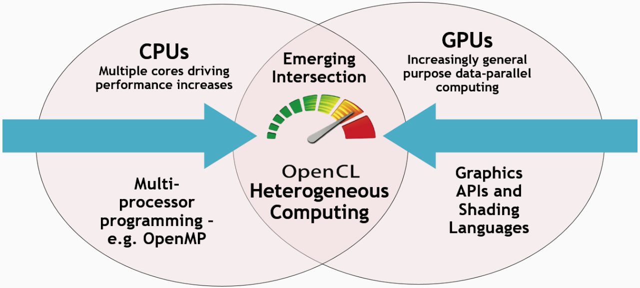

Figure 4: GPU vs CPU ©*Khronos Group, 2013*

GPU vs CPU

- Matrix multiplication performance
- Structure of Arrays vs Array of Structures

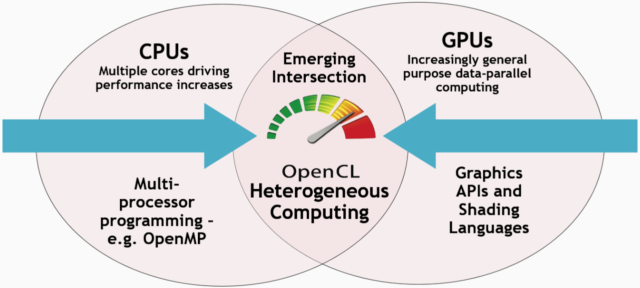

Figure 4: GPU vs CPU ©*Khronos Group, 2013*

Host & Device Code

Host code using C++ Bindings

```
cl::make_kernel<cl::Buffer,
cl::Buffer, cl::Buffer,
<int> vadd(program, "vadd");

d_a = cl::Buffer(context,
h_a.begin(), h_a.end(), true);

vadd(
     cl::EnqueueArgs(queue,
        cl::NDRange(count)),
     d_a,
     count);
```

Device code (Kernel)

```
__kernel void vadd(
   __global float* a,
   __global float* b,
   __global float* c)
{
   int i = get_global_id(0);
   c[i] = a[i] + b[i];
}
```

Statistical Algorithms

- Measures of Location
- Measures of Spread
- Measures of Shape
- Multivariate Statistics

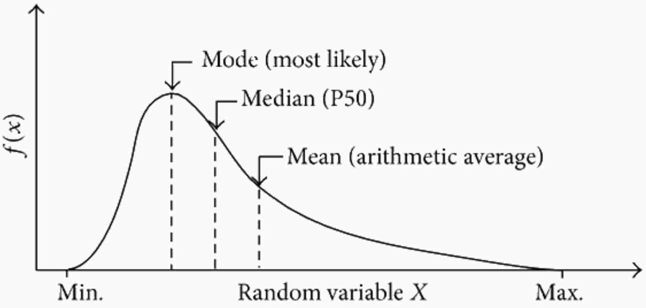

Figure 5: Central Tendency

Summary Statistics

- Measures of Location
- Measures of Spread
- Measures of Shape
- Multivariate Statistics

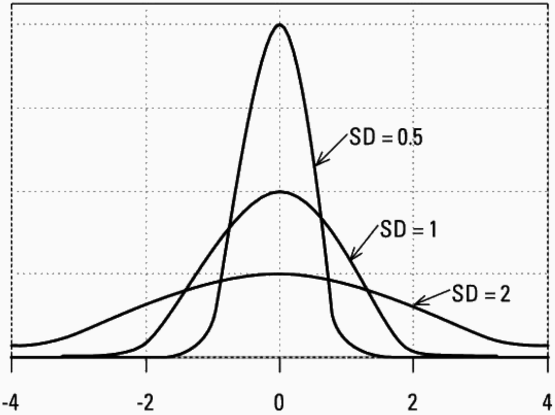

Figure 6: Standard Deviation

- Measures of Location
- Measures of Spread
- Measures of Shape
- Multivariate Statistics

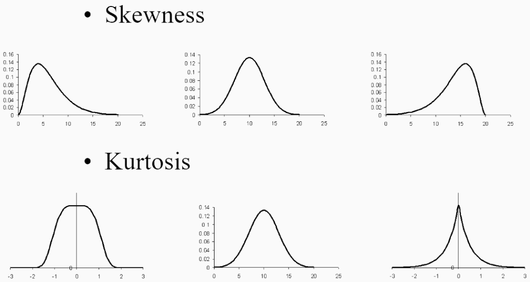

Figure 7: Skewness and Kurtosis

Summary Statistics

- Measures of Location
- Measures of Spread
- Measures of Shape
- Multivariate Statistics

Assumption: We are dealing with only one variable

Fundamental Values

The minimal statistical model from which all relevant summary statistics can be derived:

- Sample size
- Moment 1
- Moment 2
- Moment 3
- Moment 4
- Rank
- Min/Max
- Histogram

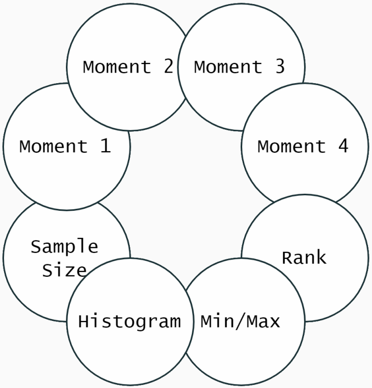

Figure 8: Minimal Statistical Model

Derived Values

From the fundamental values, we can derive all widely used summary statistics

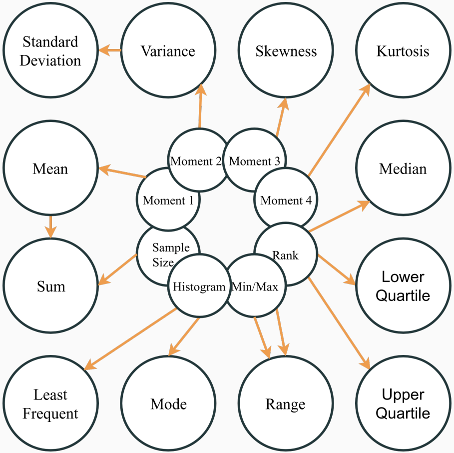

Figure 9: Extended Statistical Model

On-line Algorithms

- Sample Size - Count on host
- Moments - Equations on Right
- Rank (Median, etc) - Selection Algorithm
- Min/Max - Per-element update
- Histogram - Frequency Analysis

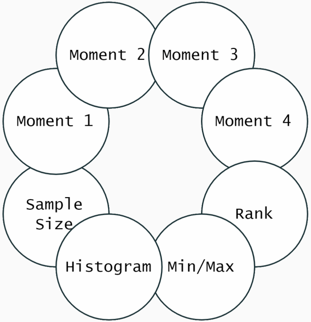

Figure 10: Minimal Statistical Model

On-line Algorithms

- Sample Size - Count on host
- Moments - Equations on Right
- Rank (Median, etc) - Selection Algorithm
- Min/Max - Per-element update
- Histogram - Frequency Analysis

As we know the size of the arrays that we are sending to the devices, it is useless to count the data points using OpenCL

On-line Algorithms

- Sample Size - Count on host
- Moments - Equations on Right
- Rank (Median, etc) - Selection Algorithm
- Min/Max - Per-element update
- Histogram - Frequency Analysis

$$
\begin{aligned}
\delta &= x - m \\
m' &= m + \frac{\delta}{n} \\
M_2' &= M_2 + \delta^2 \frac{n-1}{n} \\
M_3' &= M_3 + \delta^3 \frac{(n-1)(n-2)}{n^2} - \frac{3\delta M_2}{n} \\
M_4' &= M_4 + \frac{\delta^4 (n-1)(n^2-3n+3)}{n^3} + \frac{6\delta^2 M_2}{n^2} - \frac{4\delta M_3}{n}
\end{aligned}
$$

Figure 11: Moment Equations [Pébay, 2008]

On-line Algorithms

- Sample Size - Count on host
- Moments - Equations on Right
- Rank (Median, etc) - Selection Algorithm
- Min/Max - Per-element update
- Histogram - Frequency Analysis

k^{th} smallest element in a stream - partial sorting

On-line Algorithms

- Sample Size - Count on host
- Moments - Equations on Right
- Rank (Median, etc) - Selection Algorithm
- Min/Max - Per-element update
- Histogram - Frequency Analysis

Compare each data point to a ‘running min/max’ and update

On-line Algorithms

- Sample Size - Count on host
- Moments - Equations on Right
- Rank (Median, etc) - Selection Algorithm
- Min/Max - Per-element update
- Histogram - Frequency Analysis

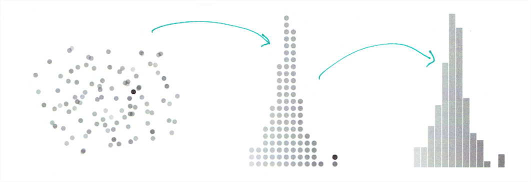

Figure 12: Histogram

Our Implementation (So Far)

Sequential

We can:

- Recalculate the base statistical values after every data-point
- Could possibly move to recalculating in chunks (possibly target cache size)

Three types of statistics to calculate:

- Minimum, maximum, and sample size are trivial
- Histogram (mode, etc.) and Rank (quartiles, median) use a HashMap
- Moments (mean, etc.) recalculation use weighted average:

$$\frac{numValuesIn(a) \times a + numValuesIn(b) \times b}{numValuesIn(a) + numValuesIn(b)}$$

Sequential: Results

Ran 3 times on an i7-7500U with 8GB RAM and a MX300 SSD

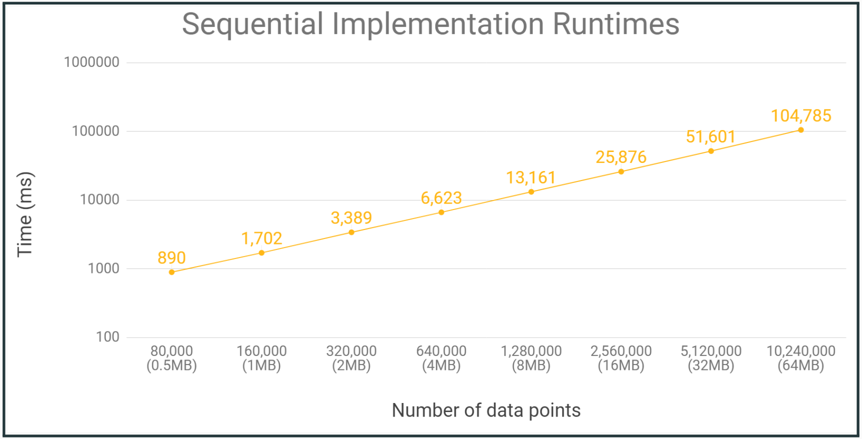

Figure 13: Sequential Runtimes

Sequential: Big Data

To really test this we tried a much bigger data set

- File-size: 4.2GiB
- Runtime: 37:32 (m:s)
- Data-points: 655,360,000
- Only run once (no averaging)

Sequential: What we learned

- Linear runtime (each data-point is processed once) as expected

Sequential: What we learned

- Linear runtime (each data-point is processed once) as expected
- Can process hundreds of millions of data points

Sequential: What we learned

- Linear runtime (each data-point is processed once) as expected
- Can process hundreds of millions of data points
- Reasonable time-frame but still slow

Sequential: What we learned

- Linear runtime (each data-point is processed once) as expected
- Can process hundreds of millions of data points
- Reasonable time-frame but still slow
- The L3 cache being loaded can be seen

Parallel Implementation

Calculation via reduction:

- Each work item is calculating on small number of points
- Each work group reduces results of the work items
- The final result is a reduction of the work groups

So far we've found GPUs can really improve performance

Parallel & Problems

Here are the problems we found during implementation:

- Reduction is hard in OpenCL (truly)

Parallel & Problems

Here are the problems we found during implementation:

- Reduction is hard in OpenCL (truly)
- Memory needs to be explicitly shared

Parallel & Problems

Here are the problems we found during implementation:

- Reduction is hard in OpenCL (truly)
- Memory needs to be explicitly shared
- Synchronising work groups

Parallel & Problems

Here are the problems we found during implementation:

- Reduction is hard in OpenCL (truly)
- Memory needs to be explicitly shared
- Synchronising work groups
- Choosing work group and work item size

Parallel & Problems

Here are the problems we found during implementation:

- Reduction is hard in OpenCL (truly)
- Memory needs to be explicitly shared
- Synchronising work groups
- Choosing work group and work item size
- Can't optimise too hard

Future Work

Optimisation

- Efficient Memory Access
- Local Memory
- Work Item Setup
- Occupancy

- Ideally we want all work items to access their data points at the same time, improving concurrency
- Structure of arrays vs array of structures

Optimisation

- Efficient Memory Access
- Local Memory
- Work Item Setup
- Occupancy

The contradiction:

- Supposed to hold data reused by ALL work items to minimise memory movements $\Rightarrow$ performance
- But CPUs don't have special hardware and modern GPU caches can give the same effect

Optimisation

- Efficient Memory Access
- Local Memory
- Work Item Setup
- Occupancy

The optimal amount of work items per work group changes based on:

- the data set and
- device architecture | GPU vs CPU (again)

Optimisation

- Efficient Memory Access
- Local Memory
- Work Item Setup
- Occupancy

Need every processing element on an OpenCL device to be active, a typical “good” measurement of occupancy is over 0.5

Testing

//TODO: Test and compare the implementations

Conclusion

Summary

- OpenCL
- Summary Statistics
  - Algorithms
- Implementation
  - Problems
  - Future Optimisation

Kahoot!™

Kahoot!™Quiz:

`https://play.kahoot.it/#/?quizId=11f98885-2501-4364-bc3c-247834e9f73d`

Questions?

Backup slides

Structure of Arrays vs Array of Structures

```
struct { float x, y, z, a; } Point;
```

- SoA: GPU, Adjacent work items prefer adjacent memory because of Memory Cohesion
- AoS: CPU, Individual work items prefer adjacent memory because of Cache Hierarchies

Figure 14: Structure of Arrays ©*Khronos Group, 2013*

Figure 15: Array of Structures ©*Khronos Group, 2013*

GPU vs CPU matrix multiplication

Case	MFLOPS	
	CPU	GPU
Sequential C (not OpenCL)	887.2	N/A
C(i,j) per work-item, all global	3,926.1	3,720.9
C row per work-item, all global	3,379.5	4,195.8
C row per work-item, A row private	3,385.8	8,584.3
C row per work-item, A private, B local	10,047.5	8,181.9
Block oriented approach using local	1,534.0	230,416.7

Figure 16: Different Multiplication Techniques ©*Khronos Group, 2013*

Main Concept Overview

- N-D Range
- Memory Model
- Kernel and Host

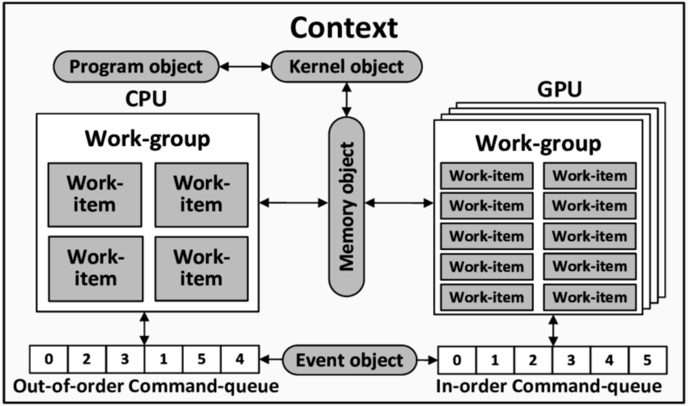

Figure 17: OpenCL Execution Model [Stone et al., 2010]

Terminology

- Host - Program managing the compute devices
- Kernel - Instructions for the compute devices
- Work Item - An element of the problem
- Work Group - Consists of work items
- N-D Range - Organisation of kernel into N-Dimensional arrays of work groups

References i

Hong, J. H., Park, J. Y., and Chung, K. S. (2016).
Parallel LDPC decoding on a heterogeneous platform using opencl.
KSII Transactions on Internet and Information Systems, 10(6):2648–2668.

Pébay, P. (2008).
Formulas for Robust, One-Pass Parallel Computation of Covariances and Arbitrary-Order Statistical Moments.
Sandia Report, SAND2008-6(September):1–18.

Stone, J. E., Gohara, D., and Shi, G. (2010).
OpenCL: A parallel programming standard for heterogeneous computing systems.
Computing in Science and Engineering, 12(3):66–72.