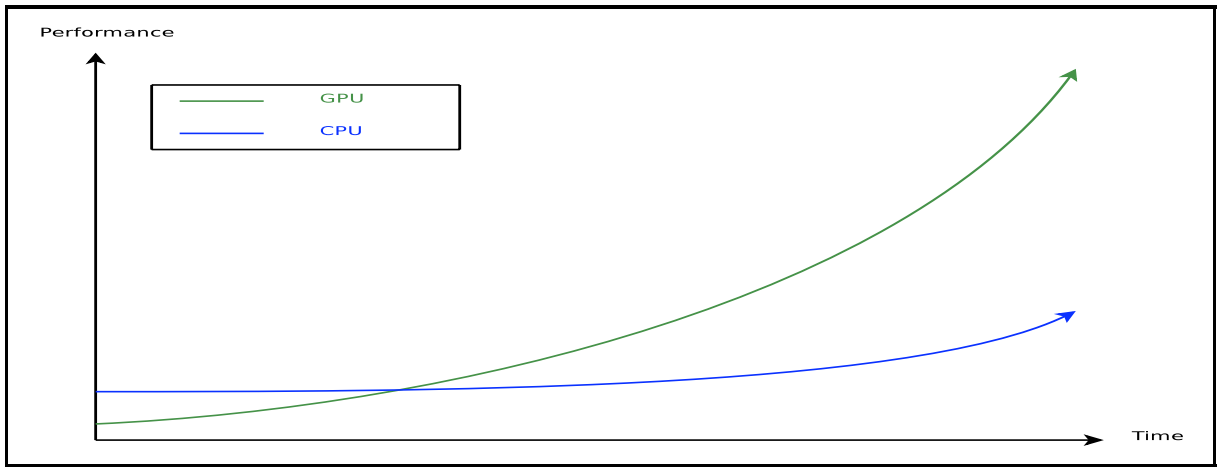
High-Performance Computing
By Advanced Stream Processing Using GPU

Sean LEE

seanl@cse.unsw.edu.au

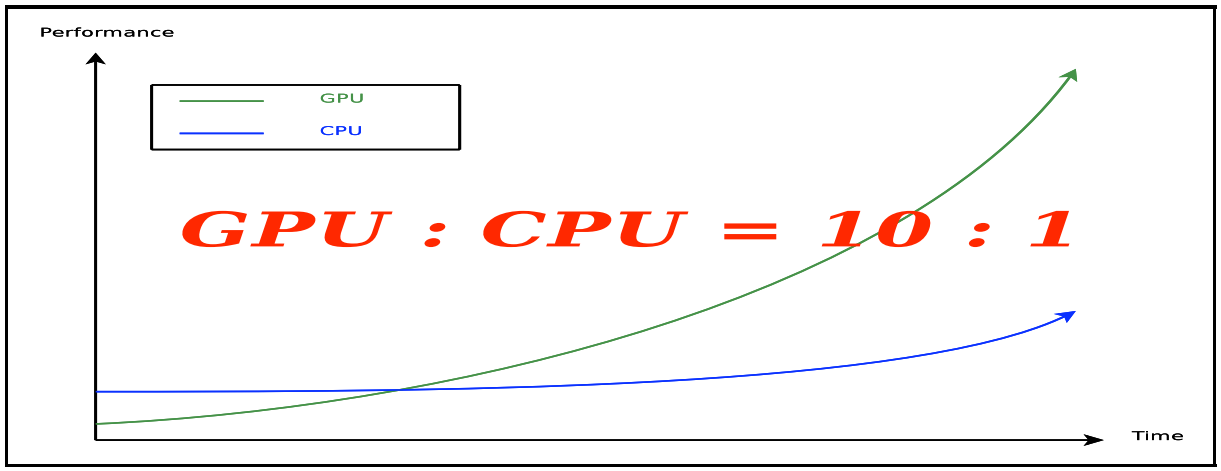
Programming Languages and Systems
School of Computer Science and Engineering
University of New South Wales

BACKGROUND



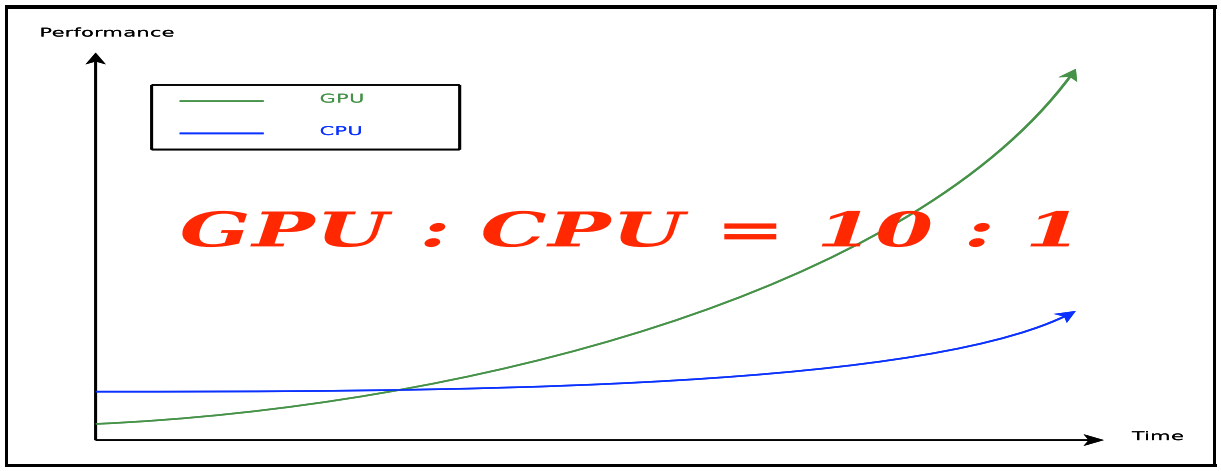
performance growth of GPUs and CPUs

BACKGROUND

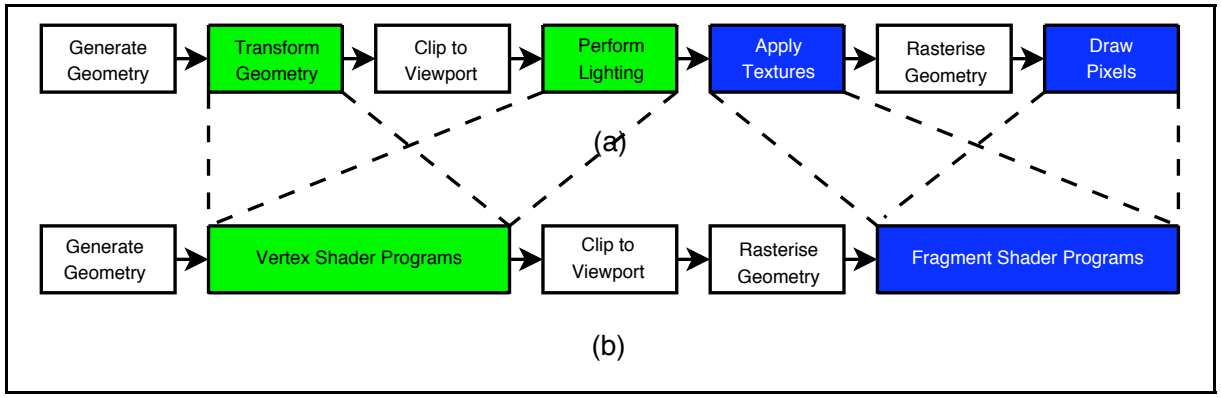


performance growth of GPUs and CPUs

BACKGROUND



performance growth of GPUs and CPUs



(a) fixed non-programmable pipelines and (b) programmable pipelines

WHAT DO I WANT TO DO WITH GPU?

What do I want to do with GPU?

NDP (Nested Data Parallelism) on GPU.

WHAT DO I WANT TO DO WITH GPU?

NDP (Nested Data Parallelism) on GPU.

Why?

- GPUs cannot handle nested or irregular data structure such as sparse matrices and nested arrays.
  Their data structure is texture-based.
- There are algorithms that can be efficiently implemented only with nested data structure.
- Manual transformation of nested data structure is error-prone and a tedious task.

What do I want to do with GPU?

NDP (Nested Data Parallelism) on GPU.

In other words:

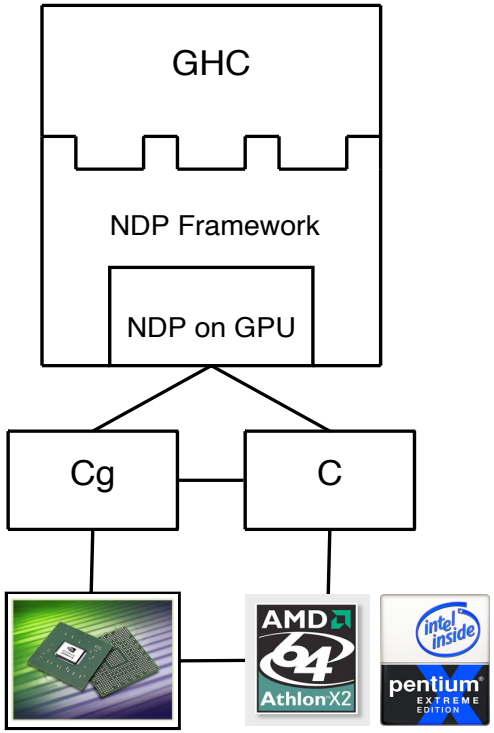
- Enhance the programmability of graphics hardware to the state where:
  - ➜ the *irregular data processing* is supported,
  - ➜ the compiler distinguishes the code to run on GPUs and the code to run on CPUs (*uniform approach*), and
  - ➜ the *hardware virtualisation* is enabled.
- Smooth integration of the enhanced programmability into the graphical applications as well as general-purpose computations.
- Achieve high-performance stream processing in low cost.

What do I want to do with GPU?

Other Programming Systems on GPU:

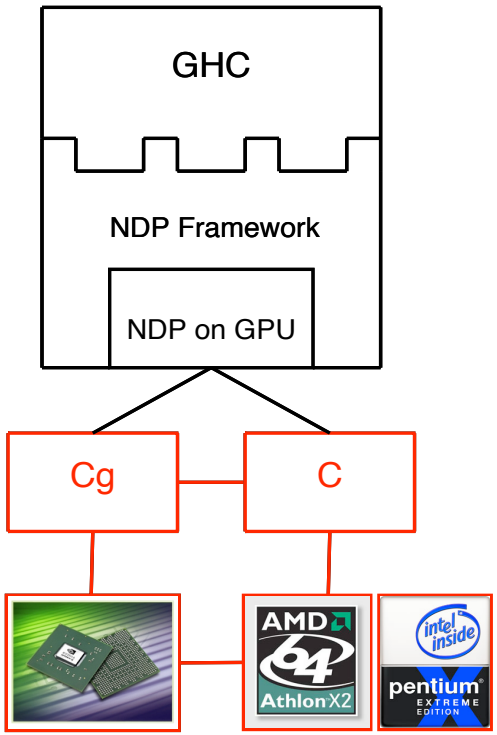
	Brook	Cg	GLSL	Sh
Nested Data Parallelism	✘	✘	✘	✘
C/C++-like syntax	✔	✔	✔	✔
Level of Abstraction	high	Low	Low	High
Standalone Compiler	✔	✔	✘	✔
Uniform Approach	✘	✘	✘	✘
Hardware Virtualisation	✔	✘	✘	✘
Irregular Data Processing Support	✘	✘	✘	✘

The Approach



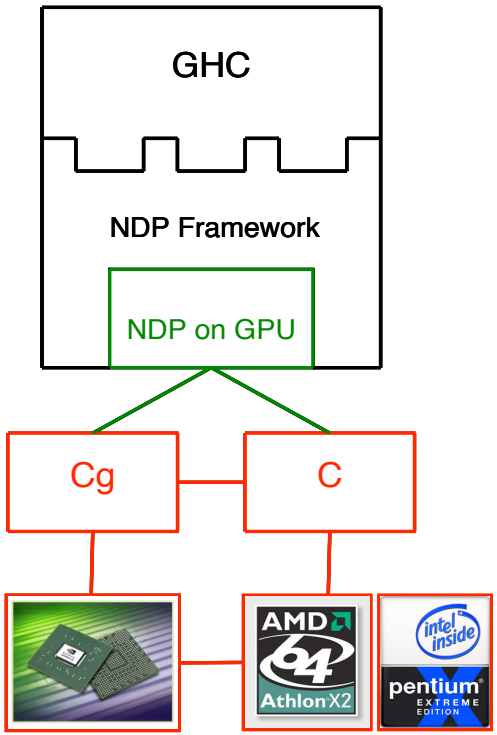
THE APPROACH

- Implement a C library for NDP on GPU. The library will include all the primitives required NDP operations on GPU.



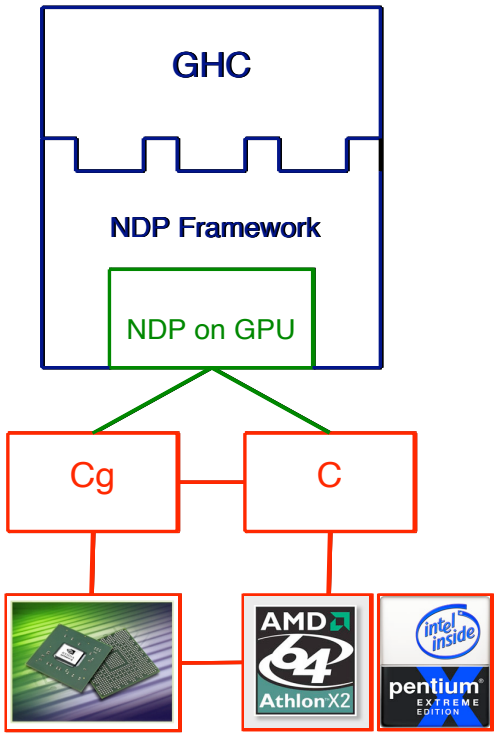
THE APPROACH

- Implement a C library for NDP on GPU. The library will include all the primitives required NDP operations on GPU.
- Develop techniques to tackle the issues such as hardware virtualisation and uniform approach.

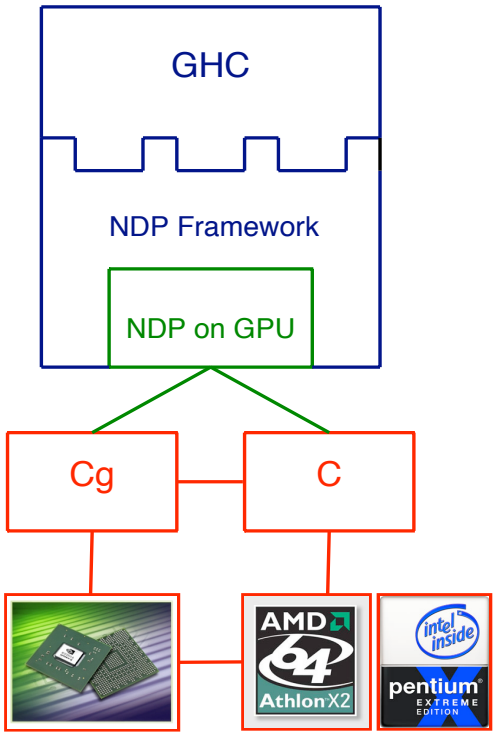


The Approach

- Implement a C library for NDP on GPU. The library will include all the primitives required NDP operations on GPU.
- Develop techniques to tackle the issues such as hardware virtualisation and uniform approach.
- The library will be hooked up to NDP framework, which is being implemented in GHC.



The Approach



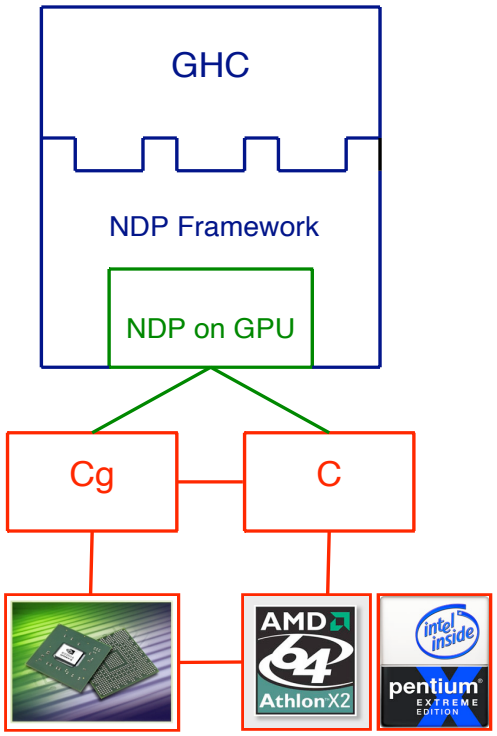
On Haskell:

```
filter ( \x->x 'mod' 2 == 1 ) input
```

On C:

```
filter (
        [[ \x->x 'mod' 2 == 1 ]],
        input, inputsize, output
);
```

The Approach



On Cg:

```
float4 main ( ... data from the host CPU ... ) : COLOR {
  ... local data declaration ...
  result.a = 0;
  index[0] = ( lW * floor ( coord.y ) + floor ( coord.x )) * s;
  for ( i = 0; i < s ; i++, index[0]++ ) {
    if ( index[0] < numElems ) {
      index[1] = index[0];
      index[2] = 0;
      while ( index[1] >= fbWidth ) {
        index[1] -= fbWidth;
        index[2]++;
      }
      V1 = ( texRECT ( elements, float2 ( index[1], index[2] ))).r;
      if ( int ( V1 ) % 2 == 1 ) {
        if ( result.a == 0 ) result.r = V1;
        else if ( result.a == 1 ) result.g = V1;
        else if ( result.a == 2 ) result.b = V1;
        result.a++;
      }
    }
  }
  return result;
}
```

Where Am I?

`libndpgpu`:

Implemented operations:

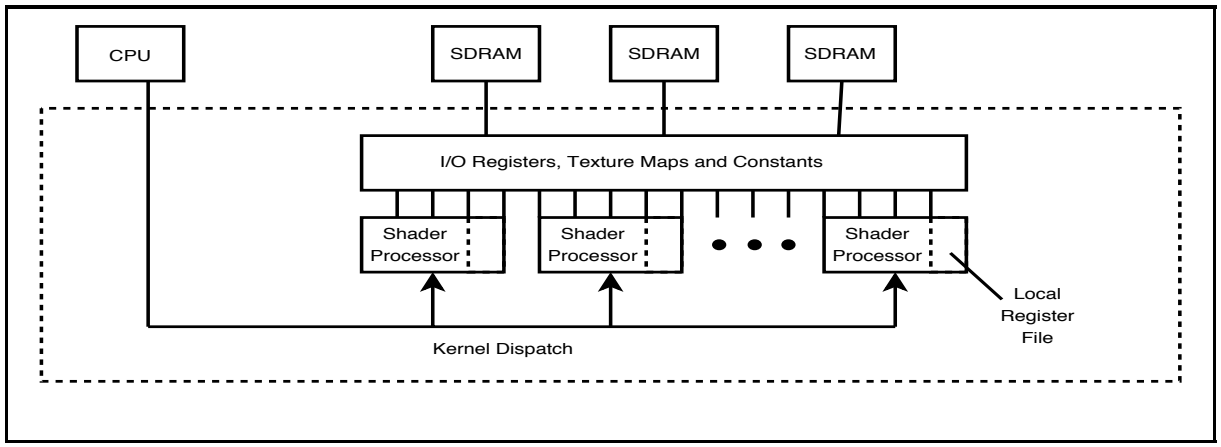
- `scanl`, `scanlS`, `scanr`, `scanrS`
- `foldl`, `foldlS`, `foldr`, `foldrS`
- `map`
- `filter`

Where Am I?

libndpgpu:

Implemented operations:

- `scanl`, `scanlS`, `scanr`, `scanrS`
- `foldl`, `foldlS`, `foldr`, `foldrS`
- `map`
- `filter`



modern GPU architecture

Where Am I?

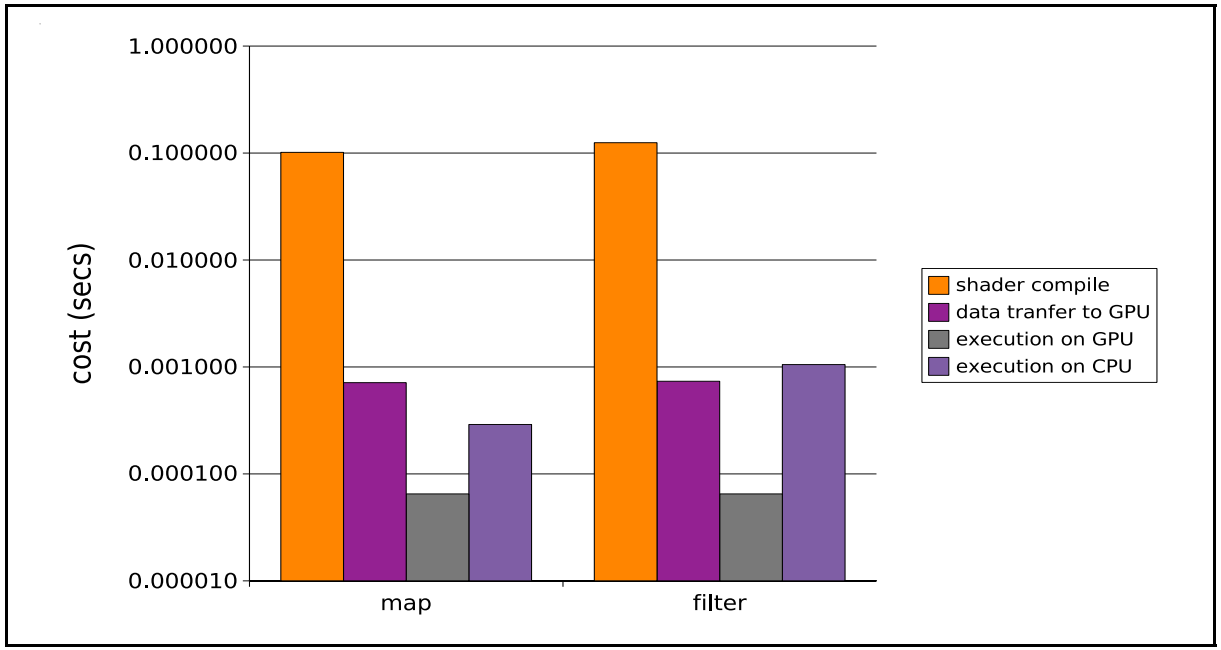
Benchmarks:

- Intel Core Duo 2.0GHz, 1 GB DDR2 RAM
- NVIDIA GeForce Go 7400 TurboCache 256MB

Where Am I?

Benchmarks:

- Intel Core Duo 2.0GHz, 1 GB DDR2 RAM
- NVIDIA GeForce Go 7400 TurboCache 256MB



CHALLENGES AND FUTURE WORKS (SHORT TERM)

Challenges and Future Works (short term)

Preprocessor:

- Profile based dynamic Cg assembly generation.
- Assembly level optimisation.
- Translation of functions passed from Haskell level to lower level as inputs to higher-order functions.

Challenges and Future Works (short term)

Preprocessor:

- Profile based dynamic Cg assembly generation.
- Assembly level optimisation.
- Translation of functions passed from Haskell level to lower level as inputs to higher-order functions.

Data organisation and transfer:

- Reduce the data tranfer cost.
- Handle arrays whose number of elements are greater than the maximum size of textures using multi-pass strategy.

Challenges and Future Works (short term)

More bulk array operations:

- permute, zip and its variations, zipWidth and its variations

Challenges and Future Works (short term)

More bulk array operations:

- permute, zip and its variations, zipWidth and its variations

Release:

- Release it as a complete library by July 2007.

QUESTIONS?