AM 148 Lecture 4

Steven Reeves

University of California, Santa Cruz

sireeves@ucsc.edu

April 16, 2020

Overview

1. Precision Support
2. Matrix Operations
   - Matrix Multiplication
   - Shared Memory MatMul
3. Stencil
   - Numerical Integration

Floats and Precision

- Floating point numbers are a representation of real numbers using rational numbers.

Floats and Precision

- Floating point numbers are a representation of real numbers using rational numbers.
- Floats have a precision type

Floats and Precision

- Floating point numbers are a representation of real numbers using rational numbers.
- Floats have a precision type
    - half precision, 16 bit floats $\sim$ 4 digits
    - single precision, 32 bit floats $\sim$ 8 digits
    - double precision, 64 bit float $\sim$ 16 digits
- Defined by the IEEE 754 standard

What are floats?

Computers at their current state know finite things.

- Real numbers can be irrational

What are floats?

Computers at their current state know finite things.

- Real numbers can be irrational
- Computers approximate real numbers using floats

What are floats?

Computers at their current state know finite things.

- Real numbers can be irrational
- Computers approximate real numbers using floats
- Floats are a combination of an exponent and mantissa

Pi by floats

To illustrate this concept, let's consider π

- $\pi \neq 3.14$

Pi by floats

To illustrate this concept, let's consider π

- $\pi \neq 3.14$
- $\pi = 3.141593653589\ldots$

$$\pi = \pi_{mach} + \mathcal{O}(\epsilon_{mach})$$

Pi by floats

To illustrate this concept, let's consider π

- $\pi \neq 3.14$
- $\pi = 3.141593653589\ldots$

$$\pi = \pi_{mach} + \mathcal{O}(\epsilon_{mach})$$

- ϵ_{mach} is the precision cutoff.

π_{mach}

- Half: $\pi_{16} = 3.141$

π_{mach}

- Half: $\pi_{16} = 3.141$
- Single: $\pi_{32} = 3.14159265$

π_{mach}

- Half: $\pi_{16} = 3.141$
- Single: $\pi_{32} = 3.14159265$
- Double: $\pi_{64} = 3.141592653589793$

Exonent and Mantissa

Formally, the computer stores floating point numbers as a mantissa and exponent, in binary. We'll use base ten:

$$3.141 = \underbrace{3141}_{\text{mantissa}} \times 10^{\overbrace{-3}^{\text{exponent}}}$$

Exonent and Mantissa

Formally, the computer stores floating point numbers as a mantissa and exponent, in binary. We'll use base ten:

$$3.141 = \underbrace{3141}_{\text{mantissa}} \times 10^{\overbrace{-3}^{\text{exponent}}}$$

There's also a bit for sign as well.

Half Precision Floats

In C++ we can use:

```
#include <half.cpp>
int main(){
        using half_float::half;
        half pi(3.141);
        std::cout<<"This is 16 bit pi!"<<pi<<std::endl;
}
```

Half Precision Floats

In C++ we can use:

```
#include <half.cpp>
int main(){
        using half_float::half;
        half pi(3.141);
        std::cout<<"This is 16 bit pi!"<<pi<<std::endl;
}
```

In CUDA we can also use the `cuda_fp16.h` header. With this we can use half natively. Note **only works with CUDA 7.5 or newer**.

Single and Double Precision

- Single and Double precision run natively in CUDA C/C++.

Single and Double Precision

- Single and Double precision run natively in CUDA C/C++.
- Single precision is most performant on most GPUs

Single and Double Precision

- Single and Double precision run natively in CUDA C/C++.
- Single precision is most performant on most GPUs
- Double precision can run on any GPU, but is only performant on some.

Precision by Nvidia Brand

	Half	Single	Double
Tesla	Pascal or Higher	All	1/2 of Single
Geforce	Not Performant	All	1/32 of Single
Quadro	Not Performant	All	1/32 of Single
Titan	Volta	All	Some Architectures

- Matrix Addtion?

- Matrix Addtion?
  Done!
- Matrix Transpose?

- Matrix Addtion?
  Done!
- Matrix Transpose?
  Homework!
- Matrix Multiplication?

- Matrix Addtion?
  Done!
- Matrix Transpose?
  Homework!
- Matrix Multiplication?
  This chapter!

Matrix Multiplication

Given two matrices A, B
$\mathbf{A} \in \mathbb{R}^{N \times M}$
$\mathbf{B} \in \mathbb{R}^{M \times L}$ then

$$\mathbf{C} = \mathbf{AB}$$

and $\mathbf{C} \in \mathbb{R}^{N \times L}$.

Sequential Matrix Multiply

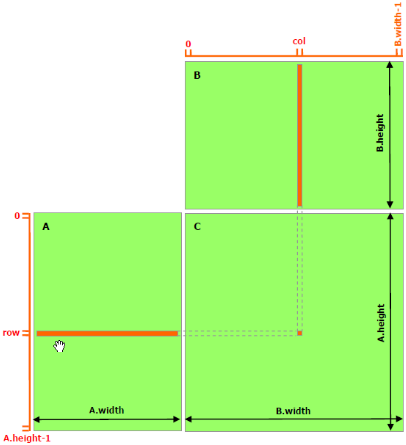
Algorithm 1: A sequential Matrix multiply

Data: $\mathbf{A}, \mathbf{B}$

Result: $\mathbf{C}$

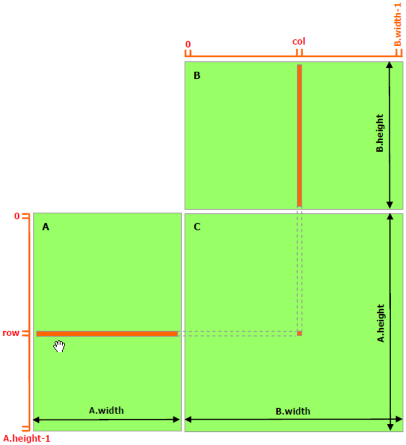
for $i = 0 \to N - 1$ **do**
    for $j = 0 \to L - 1$ **do**
        $c_{ij} = 0;$
        for $k = 0 \to M - 1$ **do**
            $c_{ij} += a_{ik} b_{kj};$
        end
    end
end

Naive Kernel



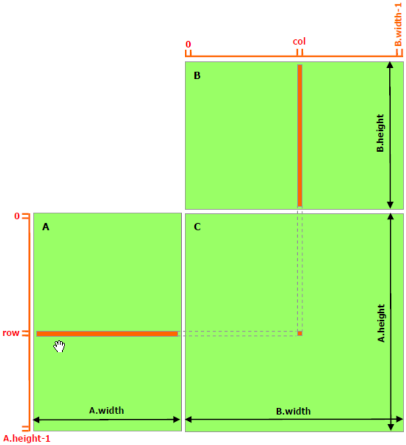
- Code Kernel

Naive Kernel



- Code Kernel
- This implementation draws from global memory significantly

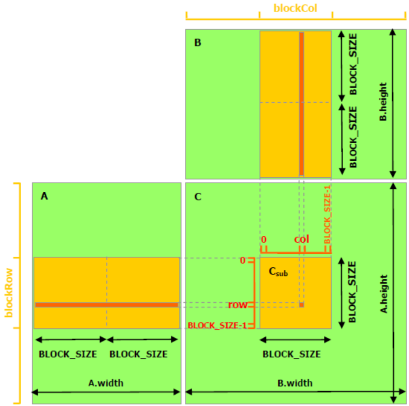
Naive Kernel



- Code Kernel
- This implementation draws from global memory significantly
- Sub-optimal on GPUs

Psycho Kernel

- Use Shared Memory
- More matrix class functions
- Tiling



Shared Memory Matrix Multiplication

- specialized `__device__` functions
- Shared Memory Kernel

Worth it?

	Serial	OpenMP	Naive CUDA	Shared Mem CUDA
$N = 32$	1.72×10^{-4}	2.102×10^{-3}	2.2×10^{-5}	1.1×10^{-5}
$N = 64$	6.15×10^{-4}	2.19×10^{-3}	2.6×10^{-5}	1.4×10^{-5}
$N = 128$	6.39×10^{-3}	3.19×10^{-3}	3.9×10^{-5}	1.6×10^{-5}
$N = 256$	5.51×10^{-2}	1.96×10^{-2}	1.43×10^{-4}	7.4×10^{-5}
$N = 512$	5.35×10^{-1}	1.58×10^{-1}	7.35×10^{-4}	2.24×10^{-4}
$N = 1024$	3.60713	1.52667	5.794×10^{-3}	1.545×10^{-3}
$N = 2048$	111.053	38.3684	4.6233×10^{-2}	1.2963×10^{-2}
$N = 4096$	—	—	3.45668×10^{-1}	7.2939×10^{-2}
$N = 8192$	—	—	4.16188	5.92996×10^{-1}

Stencil

- Class of algorithms built upon gather and map.

Stencil

- Class of algorithms built upon gather and map.
- Data is updated using a fixed set of input points, stencil

Stencil

- Class of algorithms built upon gather and map.
- Data is updated using a fixed set of input points, stencil
- Generally the stencil is much smaller than the over arching data set.

Stencil

- Class of algorithms built upon gather and map.
- Data is updated using a fixed set of input points, stencil
- Generally the stencil is much smaller than the over arching data set.
- Close to embarassingly parallel.

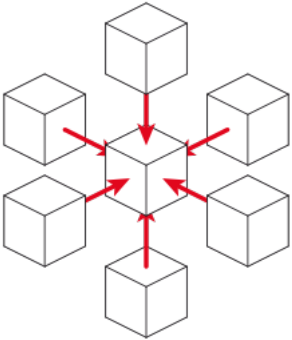


Figure: 7 point Von-Neumann Stencil

Applications of Stencil

- Numerical Partial Differential Equations

Applications of Stencil

- Numerical Partial Differential Equations
- Convolutions (Convolutional Neural Networks)

Applications of Stencil

- Numerical Partial Differential Equations
- Convolutions (Convolutional Neural Networks)
- Image Filters - "Gaussian Blur"

Applications of Stencil

- Numerical Partial Differential Equations
- Convolutions (Convolutional Neural Networks)
- Image Filters - "Gaussian Blur"
- Many more!

Numerical Integration

- Some PDEs are difficult to solve analytically

Numerical Integration

- Some PDEs are difficult to solve analytically
- Numerical Integration Schemes were developed to tackle this problem

Numerical Integration

- Some PDEs are difficult to solve analytically
- Numerical Integration Schemes were developed to tackle this problem
- Often these schemes follow the Stencil primitive.

1D Heat Equation

To illustrate the use of stencil we will solve the 1D Heat Equation

$$\frac{\partial f}{\partial t} - \kappa \frac{\partial^2 f}{\partial x^2} = 0$$

1D Heat Equation

To illustrate the use of stencil we will solve the 1D Heat Equation

$$\frac{\partial f}{\partial t} - \kappa \frac{\partial^2 f}{\partial x^2} = 0$$

- Models the heat distribution in a uniform rod
- Requires Initial Condition
- Requires Boundary Conditions

Finite Difference Schemes

$$\frac{\partial f}{\partial t} = \lim_{\delta t \to 0} \frac{f(t+\delta t) - f(t)}{\delta t} \approx \frac{f(t+\delta t) - f(t)}{\delta t}$$

Finite Difference Schemes

$$\frac{\partial f}{\partial t} = \lim_{\delta t \to 0} \frac{f(t+\delta t) - f(t)}{\delta t} \approx \frac{f(t+\delta t) - f(t)}{\delta t}$$

$$\frac{\partial^2 f}{\partial x^2} = \lim_{\delta x \to 0} \frac{f(x+\delta x) - 2f(x) + f(x-\delta x)}{\delta x^2}$$
$$\approx \frac{f(x+\delta x) - 2f(x) + f(x-\delta x)}{\delta x^2}$$

Forward Time Central Space

$$\frac{\partial f}{\partial t} - \kappa \frac{\partial^2 f}{\partial x^2} = 0$$

Forward Time Central Space

$$\frac{\partial f}{\partial t} - \kappa \frac{\partial^2 f}{\partial x^2} = 0$$

$$\Longrightarrow$$

$$\frac{f_i^{n+1} - f_i^n}{\delta t} - \kappa \frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{\delta x^2} = 0$$

Forward Time Central Space

$$\frac{\partial f}{\partial t} - \kappa \frac{\partial^2 f}{\partial x^2} = 0$$

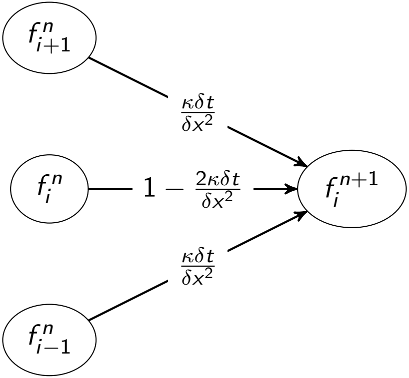
$$\implies$$

$$\frac{f_i^{n+1} - f_i^n}{\delta t} - \kappa \frac{f_{i+1}^n - 2f_i^n + f_{i-1}^n}{\delta x^2} = 0$$

$$\implies$$

$$f_i^{n+1} = f_i^n + \frac{\kappa \delta t}{\delta x^2} \left(f_{i+1}^n - 2f_i^n + f_{i-1}^n \right)$$

FTCS Stencil



Boundary Conditions

Suppose that we're modeling a uniform rod of length L. Further we suppose that the ends are perfectly insolated. In this case:

- The Heat Flux through the ends of the rod is 0

$$\left.\frac{\partial f}{\partial x}\right|_{x=-L/2,L/2} = 0$$

Boundary Conditions

Suppose that we're modeling a uniform rod of length L. Further we suppose that the ends are perfectly insolated. In this case:

- The Heat Flux through the ends of the rod is 0

$$\left.\frac{\partial f}{\partial x}\right|_{x=-L/2,L/2} = 0$$

- $f_0^n = f_1^n$
- $f_{M-1}^n = f_{M-2}^n$

Numerical Stability

There is a condition on δx and δt in order for the FTCS algorithm to remain bounded.

$$\frac{\kappa \delta t}{\delta x^2} \leq \frac{1}{2}$$

Example

- Let $L = 2$
- Using $M = 128$ data points over $x \in [-1, 1]$
- And a Guassian Heat profile

$$f(0, x) = \frac{1}{2}\exp\left(-\frac{1}{2}x^2\right)$$

Example Code