MMCP-09

C$: A High-Level System for High-Performance Graphics Processors

Andrew V. Adinetz,
Junior Research Assistant,
RCC MSU
adinetz@gmail.com

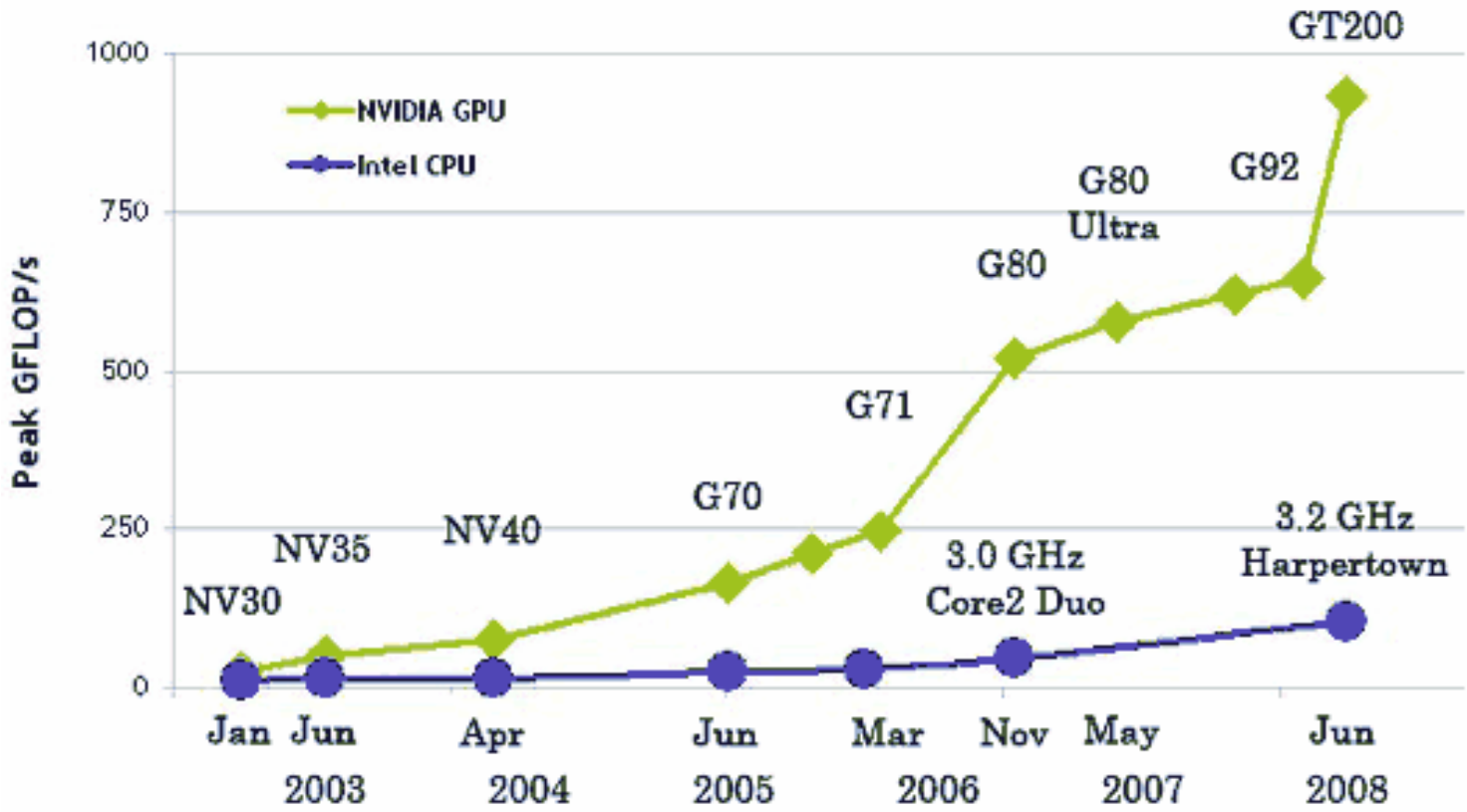
Agenda

- GPUs and how to use
- C$ Language
- C$ Runtime
- Performance Results

Agenda

- **GPUs and how to use**
- C$ Language
- C$ Runtime
- Performance Results

GPUs ...


Peak GFLOP/s
1000
750
500
250
0
NVIDIA GPU
Intel CPU
GT200
G92
G80
Ultra
G80
G71
G70
NV40
NV35
NV30
3.2 GHz
Harpertown
3.0 GHz
Core2 Duo
Jan
Jun
Apr
Jun
Mar
Nov
May
Jun
2003
2004
2005
2006
2007
2008

... and how to use

- Programming tools:
  - CUDA, OpenCL, CAL/Brook+
- Low-level:
  - Know the hardware!
  - Optimize by hand!
  - Limited portability

A tale of two architectures

	AMD 	NVidia 
Registers	128 x float4	32-64 x float
Instructions	4/5-way VLIW/SIMD	Scalar
Threads	Fully independent	Thread blocks
On-chip memory	Registry file L1, L2 Caches	16 KB shared memory/block
Memory access	Per-segment float2 indexing	Global int indexing

It's Real! (N-Body)

	Trivial	Optimized
AMD GPU (1 TFlop/s)	90 GFlop/s	700 GFlop/s
NVidia GPU (340 GFlop/s)	9.5 GFlop/s	250 GFlop/s

- => New tools wanted

Requirements

- Higher Level
- Efficiency
- Portability
  - While achieving the same efficiency
- Simplicity of integration

- => requires methods for GPU optimization

Agenda

- GPUs and how to use
- **C$ Language**
- C$ Runtime
- Performance Results

Basic Idea

- GPU Computation
  - GPU kernel = Pure function
  - Independent evaluation for all points
- GPU Program =
  - Graph of pure function applications
  - Lazy evaluation
  - For each domain point
- C$ = Java/C# + GPU constructs
  - CPU code runs in .NET

Functional Types & Objects

- **Functional type** – pure function interface
  - `float(int, int) a = new float[N, N];`
- Methods
  - f(x) — evaluation at a point
  - f.dom — get the domain
- Derived types
  - Arrays
  - Expression functions
  - Member functions

Functional Operations

- Superposition
  - `float[] a = …, b = …, c = 2 * b + 3 * b * a + exp(a);`
- Actualization
  - `float[,] a, b; …; var c = ([])(exp(a) + ln(b));`
- Reduction
  - `if(+ abs(a) < epsilon) …`

Index Variables

- **Index variable**
  - runs through a domain
  - parfor
- More complex tasks
  - Partial reduction
  - Bouding the domain
- Samples
  - `var d(j, k) = a[j, k] + sum(b[j, l] * c[l, k]);`
  - `var f[i:n, j:m] = 2.5 * i + 3.5 * (j % 2);`

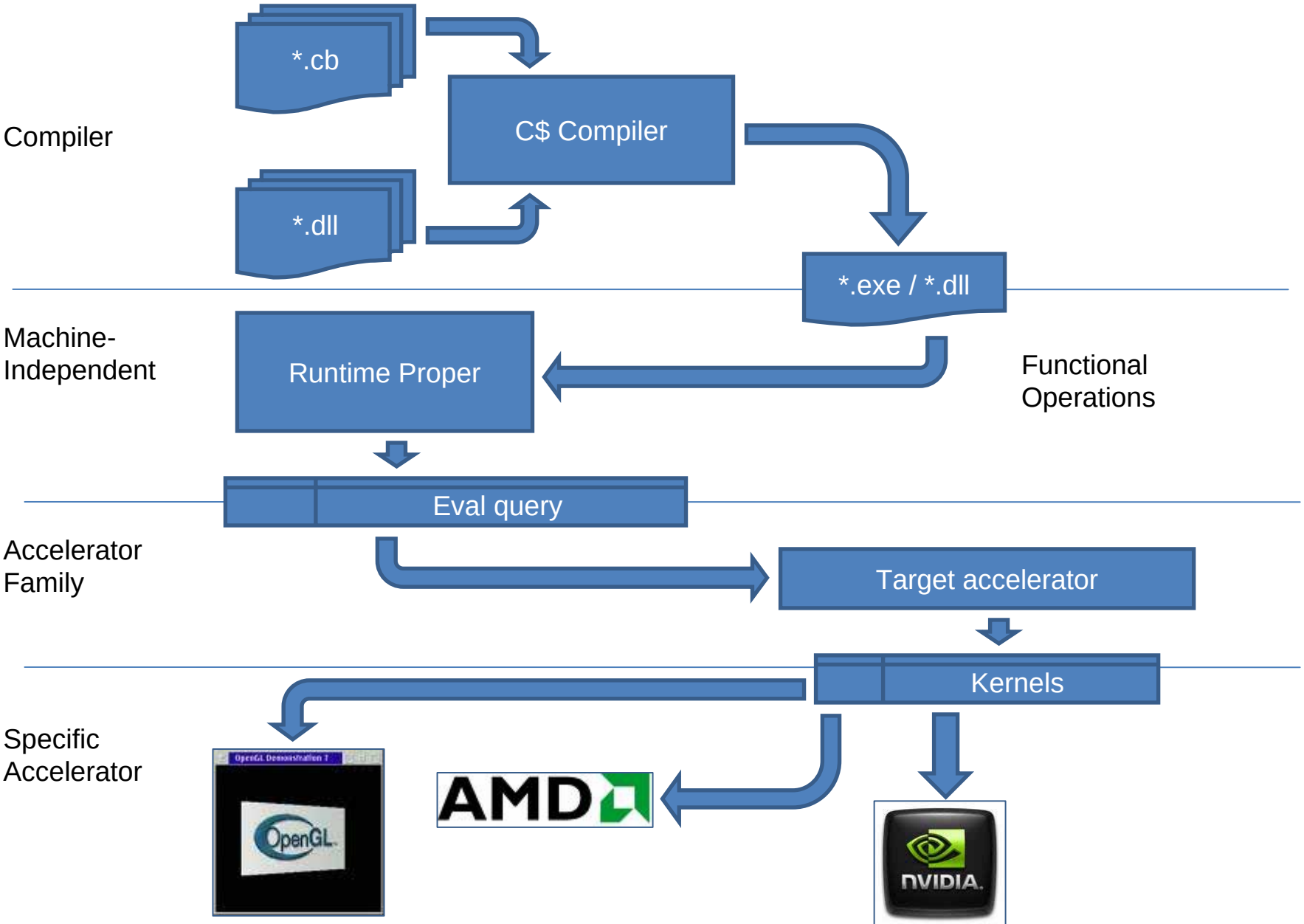
Simple matrix multiplication

```
type fmatrix = float(int, int);
…
fmatrix add(fmatrix a, fmatrix b) {return a + b;}
fmatrix mul(fmatrix a, fmatrix b) {
  fmatrix c(i, j) = + (a[i, k] * b[k, j]);
  return c;
}
…
float[,] madd(float[,] a, float[,] b, float[,] c) {
  return add(mul(a, b), c);
}
```

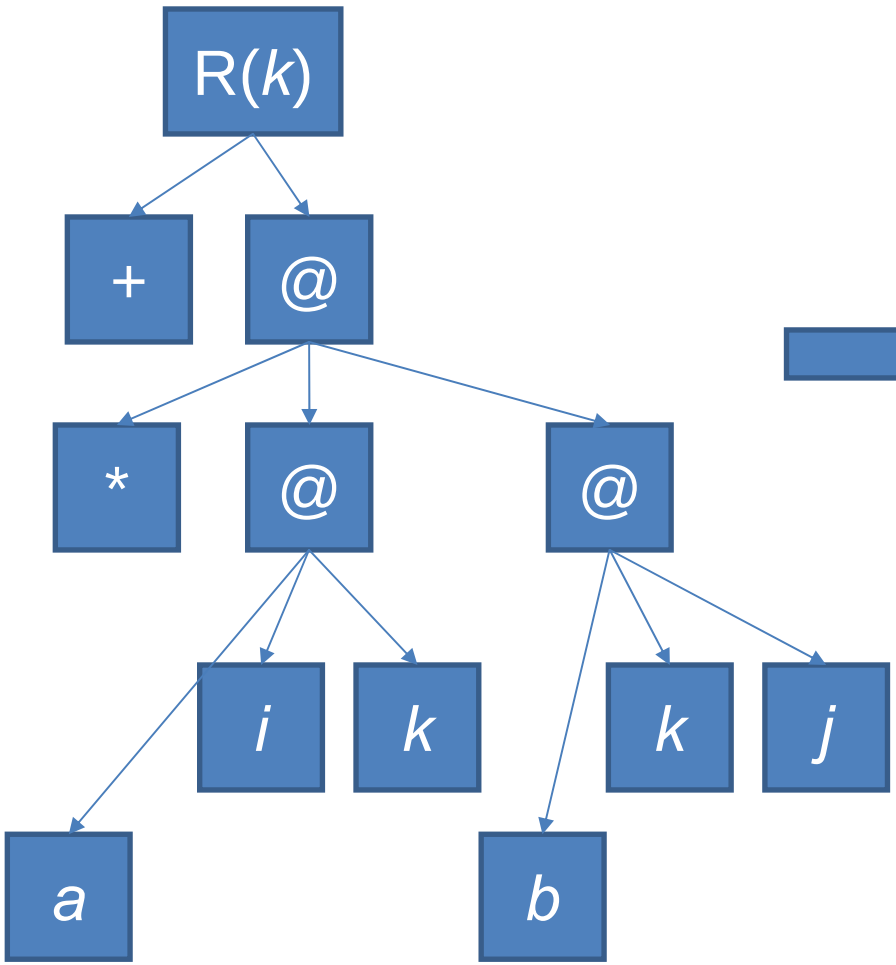
Agenda

- GPUs and how to use
- C$ Language
- **C$ Runtime**
- Performance Results

Architecture Overivew

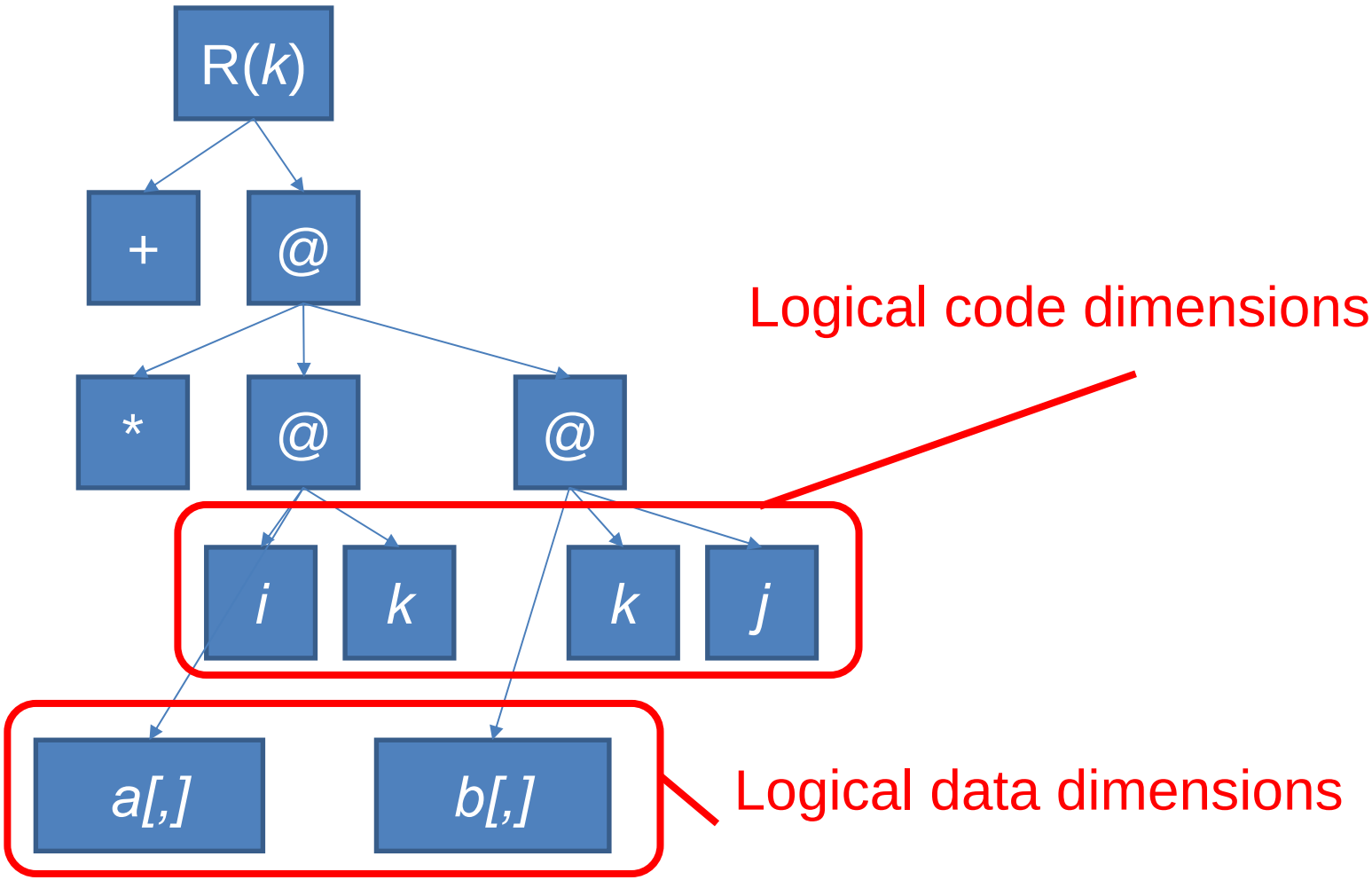


Translation to GPU



- Kernels
- Script

Logical Dimensions



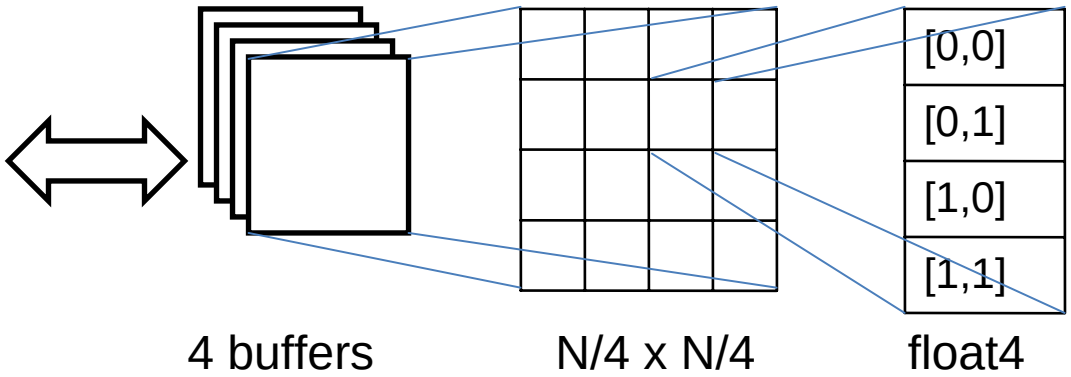
Physical dimensions

- Code:
  - SIMD/VLIW
  - Loop
  - Sequence of commands
  - Kernel domain
  - Thread block / loop on shared memory
- Data:
  - Buffer size (1D, 2D)
  - # of buffers
  - # of components (float, float4)

Mapping

[0,0]	[0,1]	[0,2]	[0,3]	…
[1,0]	[1,1]	[1,2]	[1,3]	
[2,0]	[2,1]	[2,2]	[2,3]	
[3,0]	[3,1]	[3,2]	[3,3]	
…				…

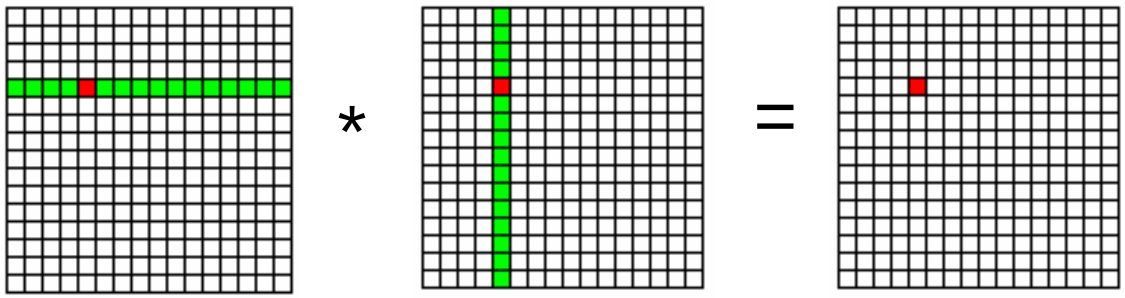
NxN data array



4 buffers N/4 x N/4 float4

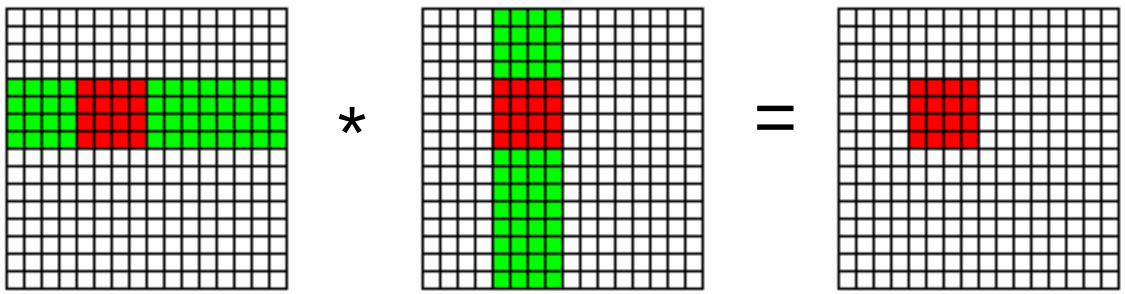
Choosing the Mapping

Matrix multiply:



MEM: $2*N^3$,
ALU: $2*N^3$
REL: 1

Block matrix multiply:



MEM: $2*N^3/K$,
ALU: $2*N^3$
REL: K

Optimization

- Find block size
  - For each index
- Limitations:
  - On-chip memory size
- Other factors:
  - Latency, # of threads
- Solving:
  - Relaxation & barrier method

Agenda

- GPUs and how to use
- C$ Language
- C$ Runtime
- **Performance Results**

Performance

	AMD HD 3870 (421 Gflop/s)	**AMD HD 4850 (1000 Gflop/s)**	**Nvidia GF8800 (340 Gflop/s)**
Matmul	157.97	**339.58**	78.8
Convolution 3*3	76.23	89.24	6.7
Convolution 5*5	92.66	143.31	2.41
Black-Scholes	230.21	273.8	204.1
Gauss 9*9	56.11	65.96	7.44
N-body	**369.70**	**746.25**	**211.60**

Acceleration

	AMD GPU	NVIDIA GPU
Matmul	**18.52**	**23.95**
Convolution 3*3	3.68	8.82
Convolution 5*5	6.03	3.35
Black-Scholes	1.52	1.02
Gauss 9*9	3.69	13.05
N-body	**5.88**	**20.02**

Acknowledgements

- Work supported by AMD Fellowship award & Intel stipend for young researchers
- Thanks to AMD & NVidia for equipment provided

Questions?