A HIGHER-LEVEL AND PORTABLE APPROACH TO GPGPU PROGRAMMING

Andrew V. Adinetz

Ph.D. Student, Moscow State University

Scientific Supervisor: Voevodin Vl. V.

adinetz@cs.msu.su

Agenda

- Why GPGPU?
- Architecture of Modern GPUs
- Programming a GPGPU
- C$ System and Language:
  - Ideas behind
  - Current work
- Future Plans

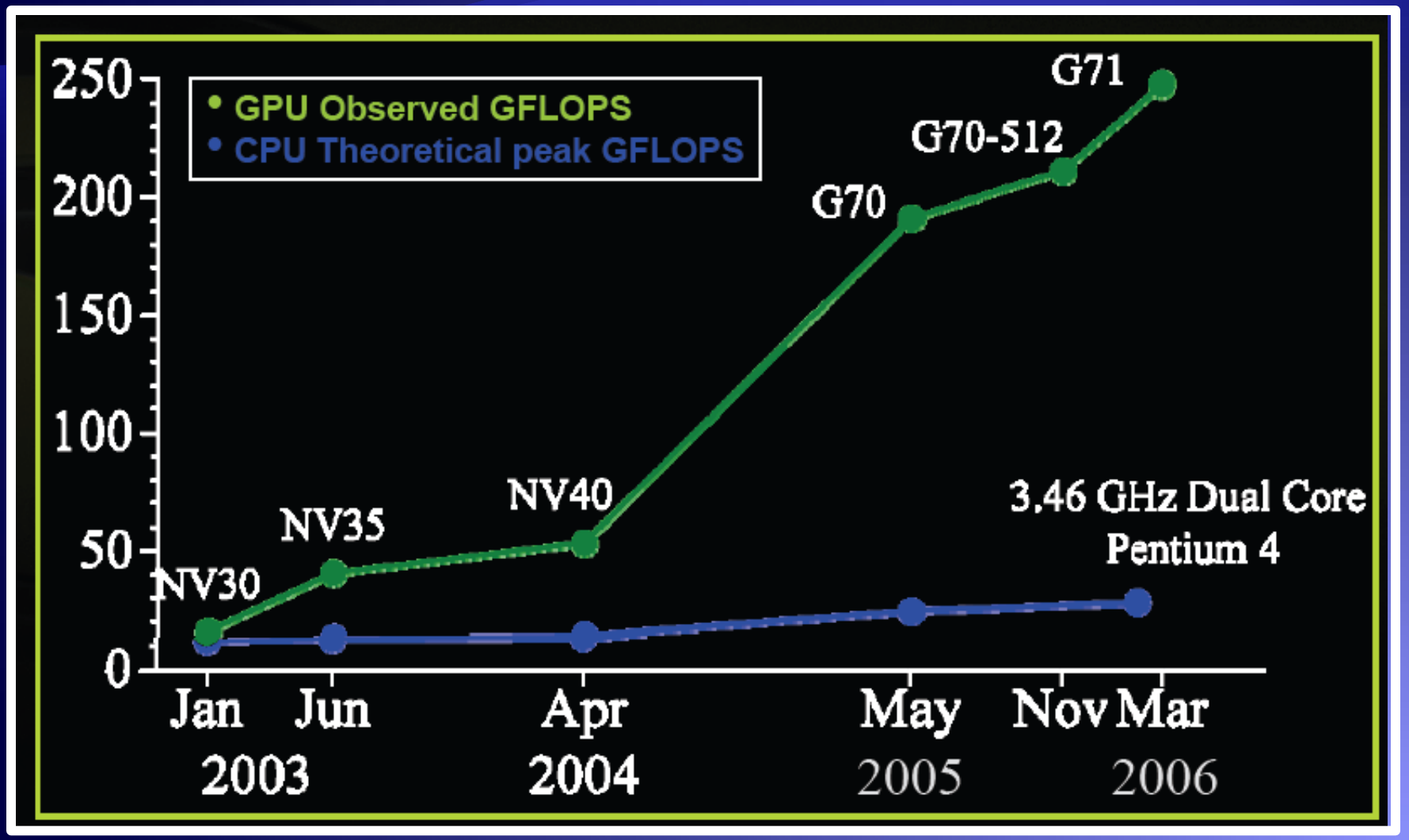
Agenda

- Why GPGPU?
- Architecture of Modern GPUs
- Programming a GPGPU
- C$ System and Language:
  - Ideas behind
  - Current work
- Future Plans

Why Graphics Processors?

- High performance(~200 GFlops)
  - On certain classes of tasks
- Availability
  - Almost in any computer
  - Idle most of the time
  - Low cost per Gflops (about $2 / Gflops)
- Trends of grows
  - Performance **doubles** each year

Comparison of CPUs and GPUs



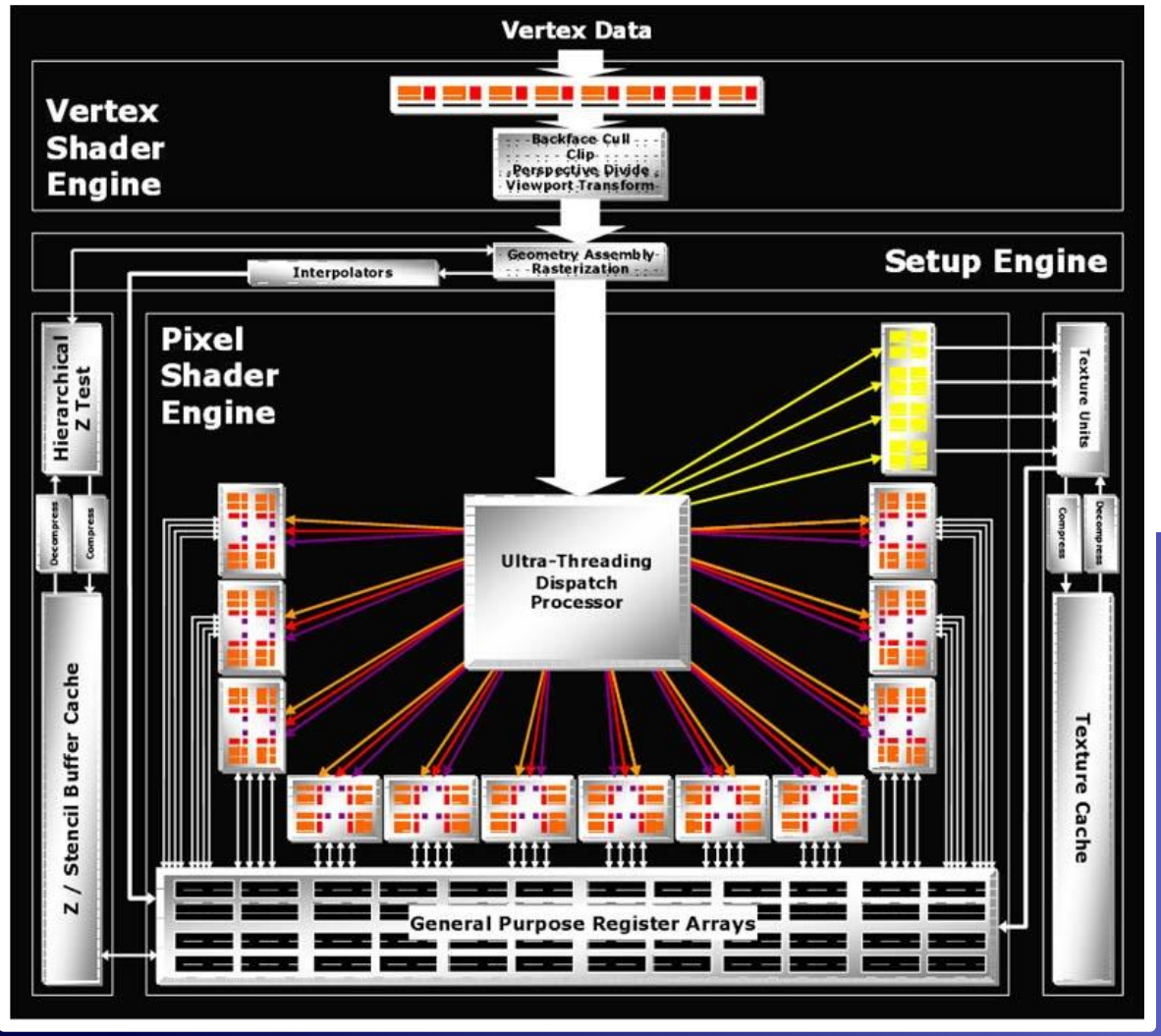
GPGPU Applications

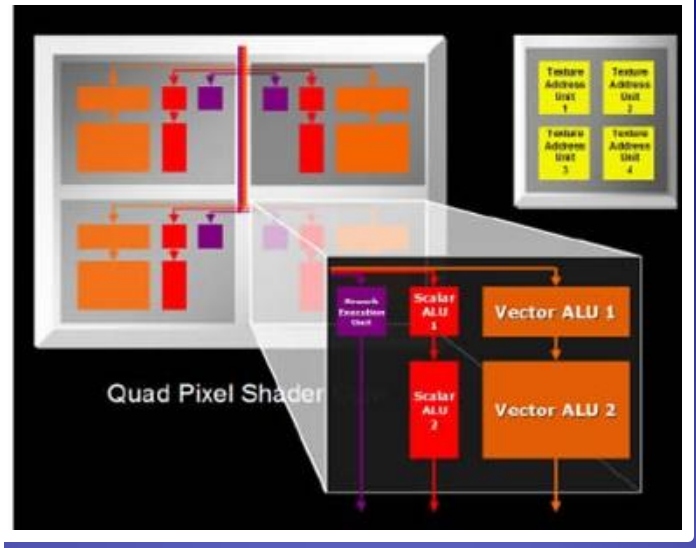
- Image & Video Processing
  - Mac OS X Core Image
- Solving of PDEs
  - Computational Fluid Dynamics
- Linear Algebra Algorithms
  - Financial & Statistical Computations
  - Risk Estimation

Agenda

- Why GPGPU?
- **Architecture of Modern GPUs**
- Programming a GPGPU
- C$ System and Language:
  - Ideas behind
  - Current work
- Future Plans

Processor Elements





Instruction Set of Processing Elements

- Arithmetic Instructions
  - Scalar & Vector
- Computations of elementary functions
- Control flow instructions
- Memory access

Agenda

- Why GPGPU?
- Architecture of Modern GPUs
- **Programming a GPGPU**
- C$ System and Language:
  - Ideas behind
  - Current work
- Future Plans

Typical GPGPU Programming

- Splitting the algorithm into kernels
  - Or shaders
- Loading data into textures
- Setting up the graphics pipeline
- Executing a shader
  - Getting output
- Repeating for each pass
- Usually done with OpenGL or DirectX

Streaming Programming

- Higher-level of abstraction
  - Streams & kernels not tied to GPU
- Portability
  - Multi-core, CELL
- Basic array operations
  - Reduction
- Libraries commercially available
  - PeakStream, RapidMind
- Still rather low-level

Agenda

- Why GPGPU?
- Architecture of Modern GPUs
- Programming a GPGPU
- **C$ System and Language:**
  - **Ideas behind**
  - Current work
- Future Plans

A GPGPU Computation is…

- Function evaluation
  - No side effects
  - Over a domain
  - Independently for each value
- Expresses operations in a more natural way
  - Non-local operations
  - Non-“one-to-one” operations

Why a Language?

- Using .NET Framework
  - Eases language development
  - Language constructs => library calls
  - Easier integration with OOP
- Convenient for programming
  - C#-like (with functional-array constructs)
  - Writing less code
  - Portability

Functions and arrays

- Array is a special case of functions
  - Lazy computations
- Small number of basic operations:
  - Superposition
  - Reduction
- Several types of functions
  - Data-only functions (arrays, maps)
  - Code-only functions (members of classes)
  - Encapsulated expressions (lazy computations)

Matrix multiplication example

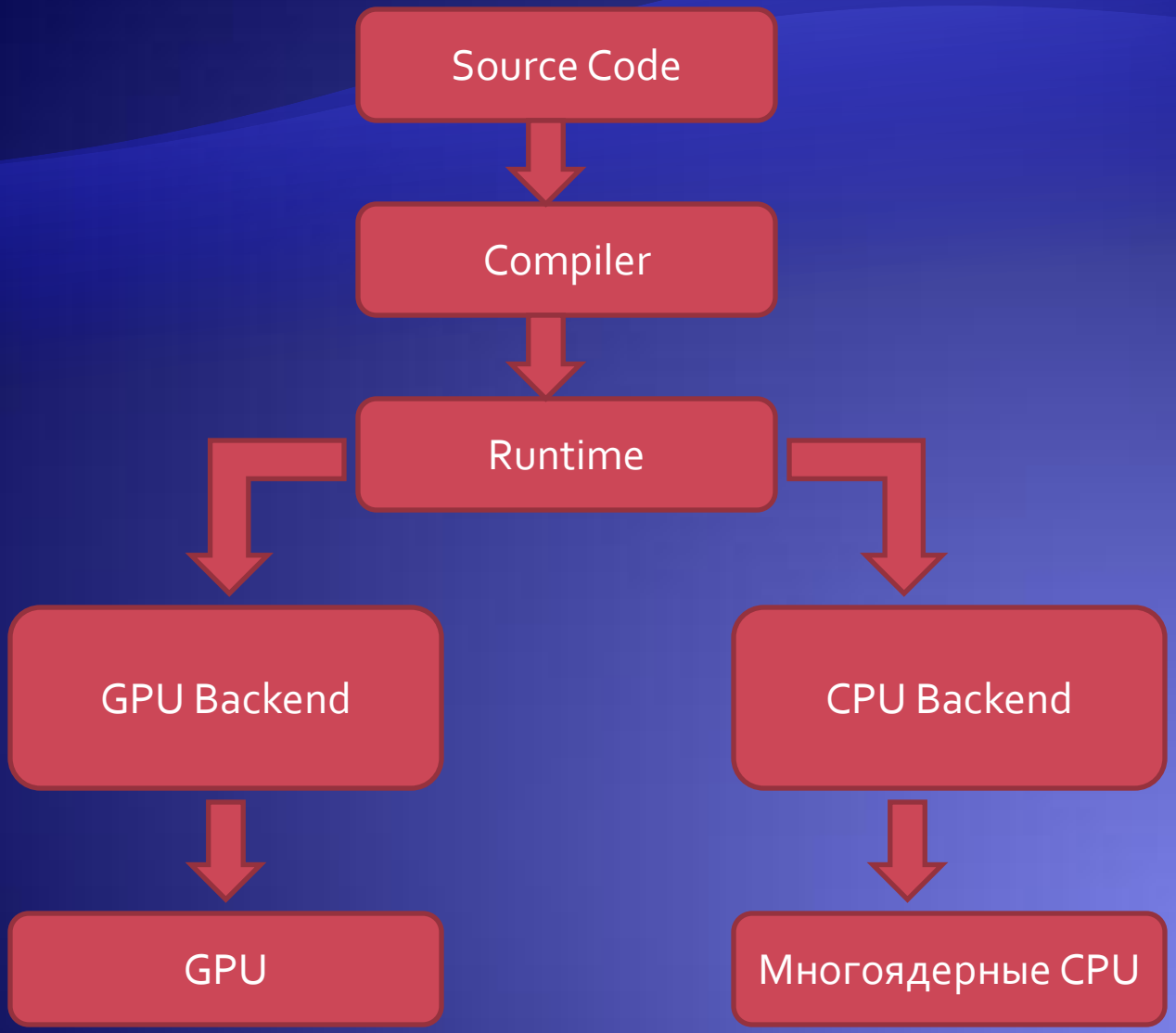
```
type matrix = float (int, int);

matrix mulMat(matrix a, matrix b) {
  var c(k, m) = sum(a(k, l) * b(l, m));
  return c;
}  // end of mulMat()

public static void main(string[] args) {
  float[int,int] a, b, c;
  a = Utils.readMatrix(args[0]);
  b = Utils.readMatrix(args[1]);
  c = mulMat(a, b);
  Utils.saveMatrix(c, args[2]);
}  // end of main()
```

Functions and objects

- Functions as interfaces
  - Inheriting from a function
  - **class** SparseMatrix : float (**int**, **int**) {…}
  - class GridFile : float3 (int, int, in) {…}

- Dot acting as superposition:
  - **final class** float3 {**float** x, y, z; … }
  - float3 (**int**) f = …;
  - f.x – is a function

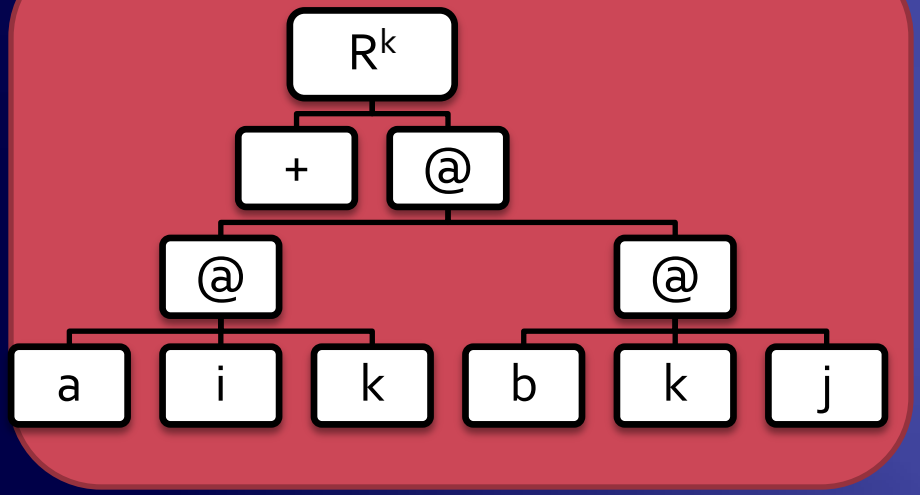
System Architecture



Example of Translation

C$:

```
float(int, int) a, b, c;
c[i, k] = sum(a[i, j] * b[j, k]);
```



GLSL Shader:

```
uniform sampler2D a, b;
uniform int l;
float4 psmain(int i, int j) : COLOR0 {
 float22 c = float22(0, 0, 0, 0);
 for(int k = 0; k < l; k++) {
  float22 a1 = float22(tex2D(a, i, k));
  float22 b1 = float22(tex2D(b, k, j));
  c += mul(a, b);
 }
 return c;
} // end of psmain()
```

Agenda

- Why GPGPU?
- Architecture of Modern GPUs
- Programming a GPGPU
- C$ System and Language:
  - Ideas behind
  - **Current work**
- **Future Plans**

Current State of the Project

- Support for array operations
  - Function Application
  - Reduction
  - No bound variables
- Backends
  - ATI DPVM (Close-To-Metal)
  - DirectX
  - Use of SIMD Instructions

Support

- C$ project is supported by ATI Fellowship award for Ph.D. Students

Future Work

- Full-Scale .NET Compiler
- Multi-Core & CELL Support
- Backend support for structured types
- Support for more complex operations

Links & Resources

- http://www.codeplex.com/cbucks/ - project site
- http://www.gpgpu.org/ - major GPGPU resource

QUESTIONS?