

IEEE Computer Society Workshop on Multi-core Computing, 16 Nov 2007

Programming Heterogeneous Cores: Lessons from GPGPU

**Veenus.A.V, Anoop Thomas
and Rajesh.R**

NeST

Outline of Talk

- **Motivation**
- **Stream Computing**
- **GPU based Stream computing**
- **Graphics Pipeline**
- **CPU/GPU analogies**
- **Programming using Graphics APIs, RapidMind**
- **Case Study.**

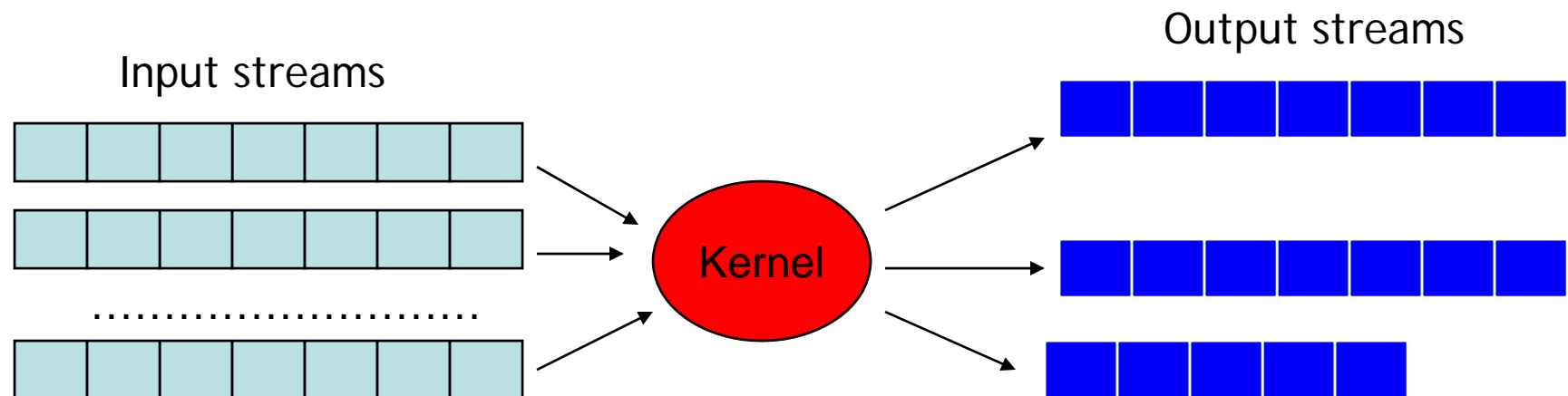
Motivation

- Modern Semiconductor technology makes *arithmetic processing and bandwidth* very expensive
- To exploit this shift in cost, a high performance computer system must exploit
 - locality, to raise the *arithmetic intensity* (the ratio of arithmetic to bandwidth) of the application
 - *Parallelism* - keep a large number of arithmetic units busy.
- Expressing an application as a *stream program* fulfills both of these requirements.
- High Arithmetic intensity requires that communication between stream elements be minimized.

The logo for NEST, consisting of the letters 'NEST' in a bold, sans-serif font.

What is stream computing

- **Simple illustration of stream computing**



Kernel is called computing operation, Output streams

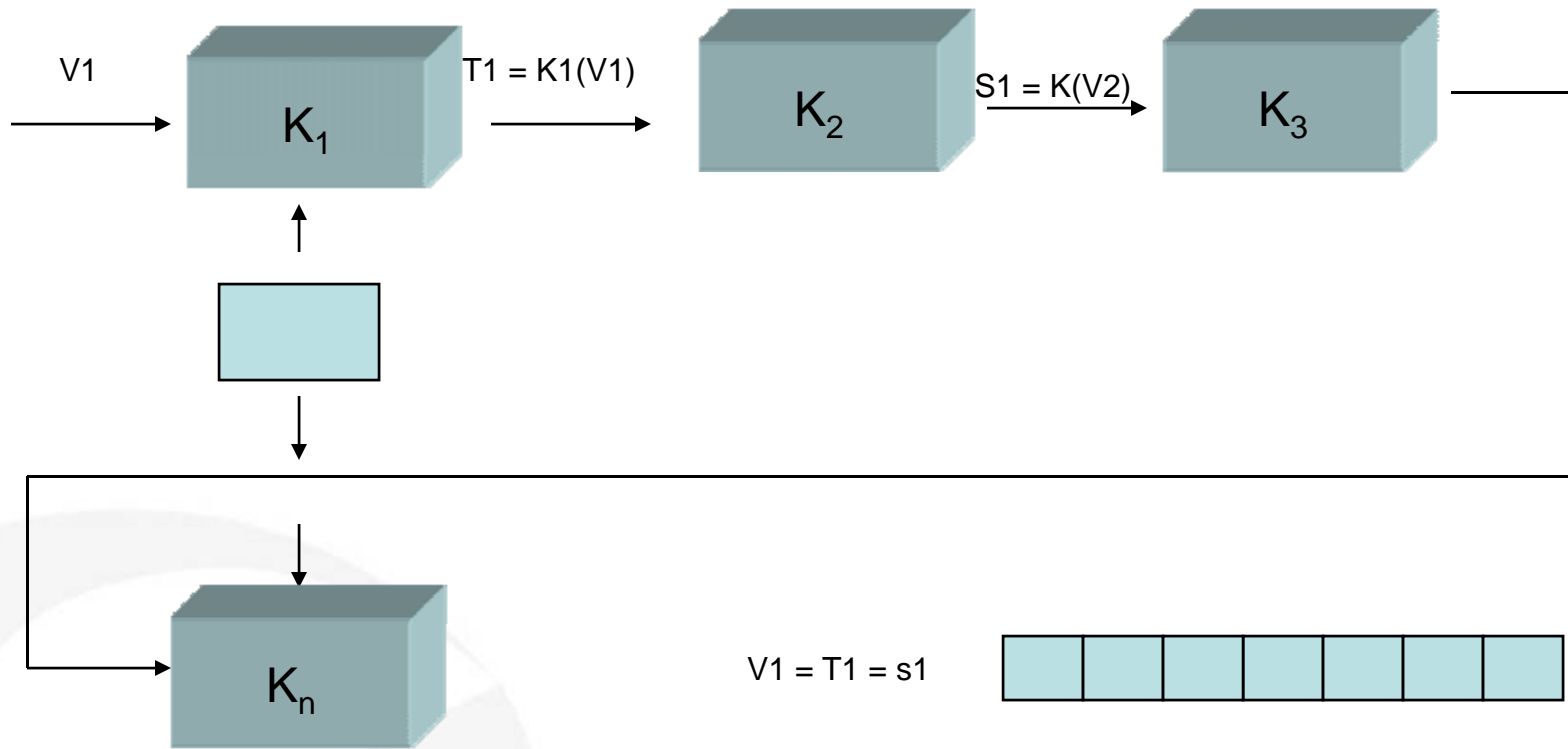
A Kernel is applied to each element of the stream in a parallel way

Stream Programming Model

- Data is represented as streams
 - Streams are ordered set of data of **same type**
 - Data type can be **simple** (stream of ints and floats) or **complex** (streams of matrices, stream of points, triangles)
- Kernels operate on entire streams
 - Taking one or more streams as inputs and produces one or more stream as out output
 - Typical use of kernel is to **evaluate a function** on each element
 - Desirable kernel operations: Expansion, Reduction, filters etc
- Kernel computation on one stream element is never dependent on computation on another element
 - Intermediate computed data is **locally** or are carefully controlled as **global references**
- Applications can be constructed by **chaining multiple kernels** together
 - Communication between the kernels becomes **explicit**, taking advantage of the data locality between Kernels inherent

Chaining the multiple kernels

- Chaining multiple kernels



NeST

Serial Programming v/s Stream Programming

CPU

Read the next instruction and
decode it
Fetch a number
Fetch another number
Add them
Put the result on c
Read the next instruction and
decode it
read the next instruction
Decode it fetch this number
fetch that number
add them put the result there

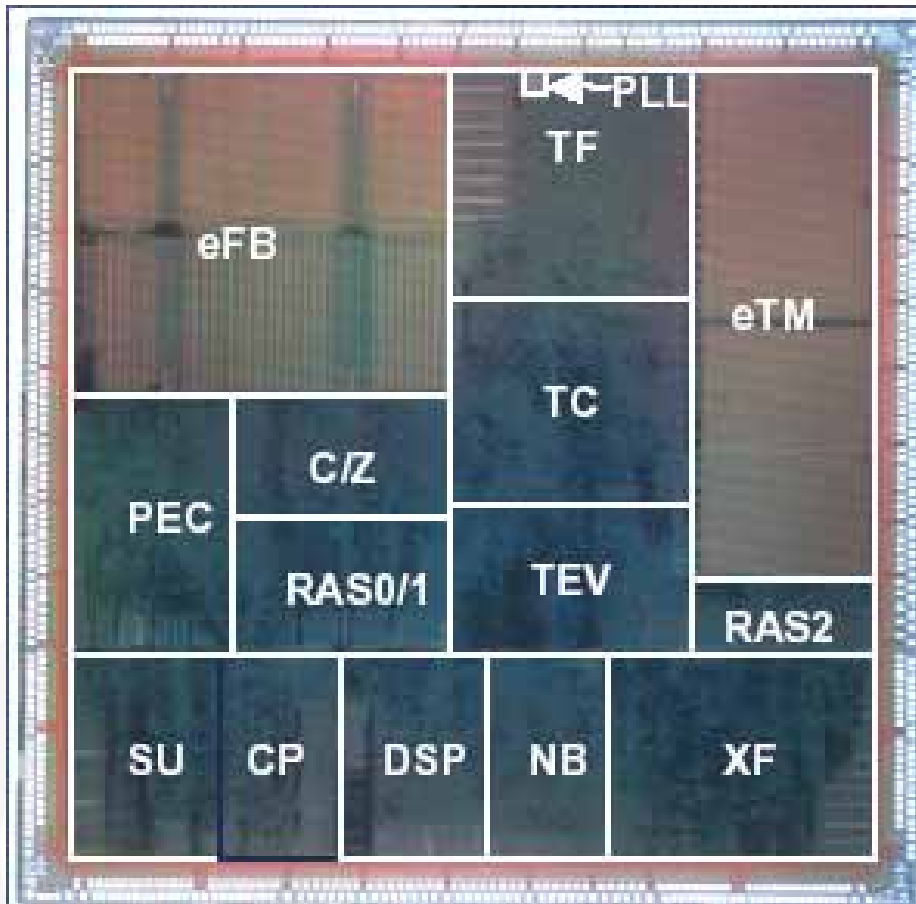
Stream processor

Read the next function and
decode it
Fetch these 10 numbers
Fetch those 10 numbers
Add them
Put the result here

Types of stream Modeling

- **Uniform Stream Modeling -**
 - Single kernel function being applied to several type of uniform type
 - **GPU** is the popular stream processor and this is going to our discussion.
- **Non-Uniform Stream Modeling.**
 - Multiple Kernel functions being applied to uniform data type

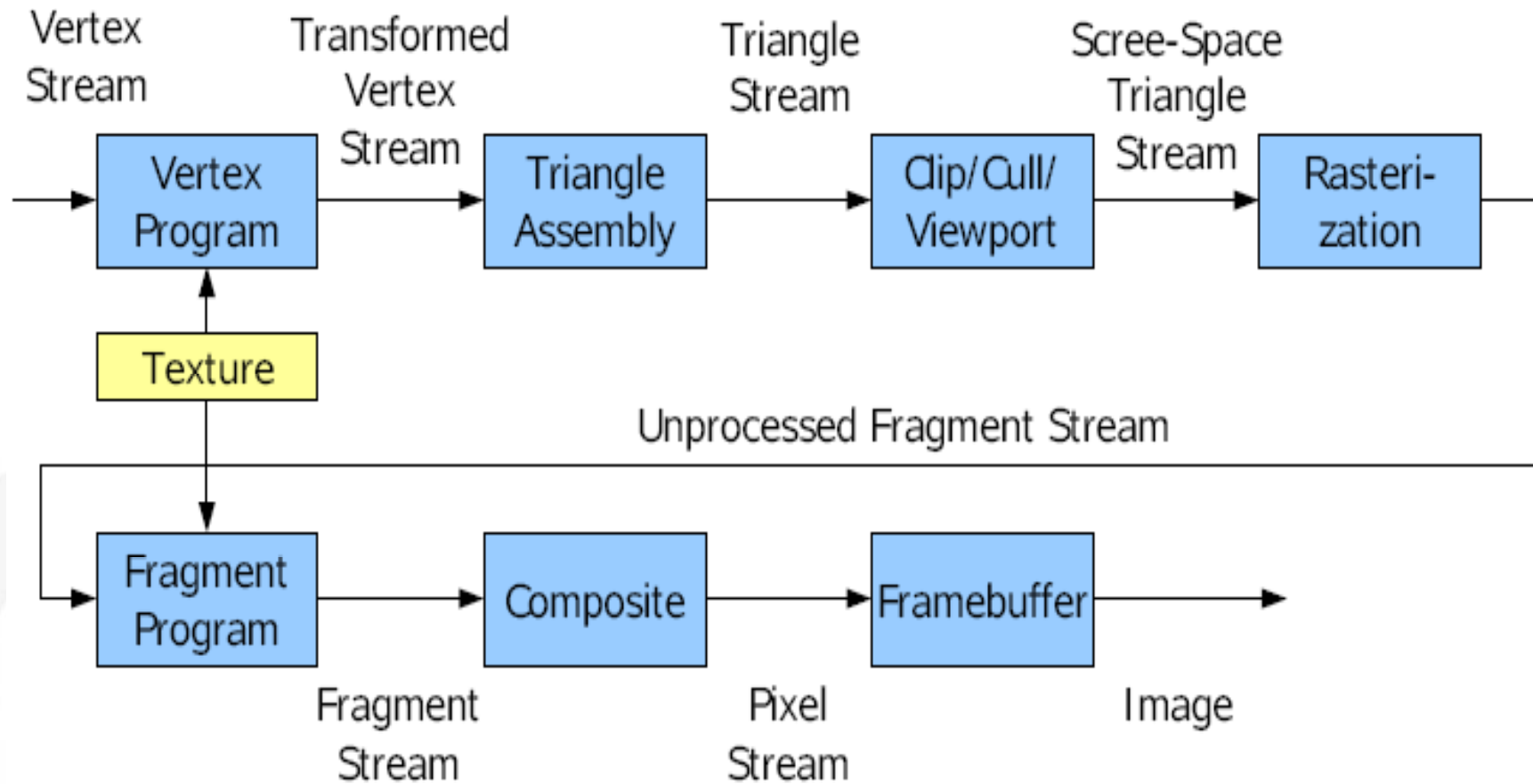
GPU: Graphics Processing Units



[ATI Flipper – 51M T]

- Dominant graphics architecture
- GPU architecture is based on Programmable Rendering Pipeline
- Specialized hardware units
- Data-parallel processing - Arithmetic processing units
- Consists of several **vertex processors cores** and **pixel processor (cores)**
- Allows efficient communication between kernels functional units implementing neighboring kernels in the graphics pipeline are adjacent on the chip

Graphics Pipeline as stream Programming Model



GPGPU (General Purpose computing on
Graphics Hardware)
How General Purpose applications motivated
from GPU

The Nest logo is a stylized, light gray graphic consisting of several overlapping curved shapes that form a partial circle or sphere. The word "Nest" is written in a bold, sans-serif font across the center of this graphic.

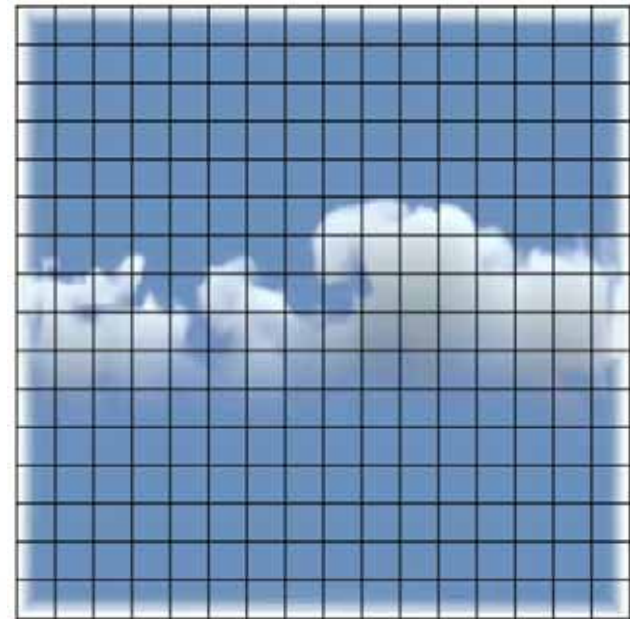
Nest

GP Motivation of GPU

- **Most of the functional units in GPUs have high arithmetic processing**
- **On Chip memory eliminates memory latency**
- **Data parallelism – Identical computations performed on large datasets (A kind of Computing on Grid)**
 - Image Processing (Pixels as streams and various operations as kernels)
 - Video Processing (Video streams)
 - Bioinformatics (genome data streams)
 - Volume Rendering (Voxels as data streams)

Example: Grid

- **Common GPGPU computation style**
 - Textures represent computational grids = streams



NeST

Computational Resource Inventory

- **Programmable parallel processors**
 - Vertex & Fragment pipelines
- **Rasterizer**
 - Mostly useful for interpolating addresses (texture coordinates) and per-vertex constants
- **Texture unit**
 - Read-only memory interface
- **Render to texture**
 - Write-only memory interface

CPU-GPU Analogies

- Use pixel shaders as computation engine
- CPU/GPU analogies
 - Data Array => Texture (Stream)
 - Memory Read => Texture Lookup
 - Loop Body => Shader Program (Kernel)
 - Memory Write => Render to Texture
- Restricted I/O : arbitrary read, limited write
- Program invocation

Fragments Program = Inner Loops

- **CPU**
 - Loop with a body inside
- **GPU**
 - Similar instructions in Fragment program
 - Applied to all elements of the stream(loop implied)
- **Parallelism**
 - No of fragment processors
 - Four vector GPU arithmetic

Code in CPU using C++

```
for (x=0; x < WIDTH; x++)
  for (y=0; y < HEIGHT; y++)
    for (i=-1; i <= 1; i++)
      for (j=-1; j <= 1; j++)
        float w = computeWeight(i,j);
        blurImage[x][y] += w * image[x+i, y+j];
```

Simple Application

Code in GPU using shader

```
float4 blurKernel( uniform samplerRECT image,
                  float2      winPos : WPOS,
                  out float4  blurImage )
```

```
{
    blurImage = float4(0,0,0,0);
```

```
    for (i=-1; i <= 1; i++) {
        for (j=-1; j <= 1; j++) {
            float2 texCoord = winPos + float2(i,j);
            float w      = computeWeight(i,j);
            blurImage += w * texRECT( image, texCoord );
        }
    }
}
```



Nest

Render To Texture = Feedback

Finish kernel before next kernel can proceed

- Needs Intermediate storage (on Texture unit)

Trivial on CPU:

- Memory can be read or written anywhere in a program

More complicated on GPU:

- Arbitrary texture read in kernel (implicit)
- Write only at end of kernel
- Fundamentally separate processes

NeST

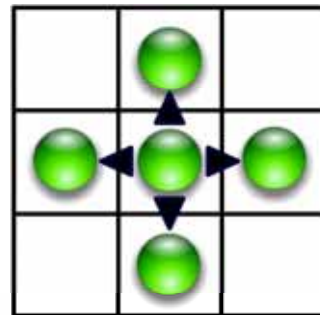
GPU techniques

- **Map:** apply a function to every element in a stream
 - Basic GPU operation
- **Reduce:** use a function to reduce a stream to a smaller stream (often 1 element)
 - Usually used for finding the maximum or sub sum
- **Scatter/gather:** indirect read and write
 - Vertex processors are capable of scatter only
 - Pixel processors are capable of gather only
- **Filter:** select a subset of elements in a stream
- **Sort:** order elements in a stream
- **Search:** find a given element, nearest neighbors, etc

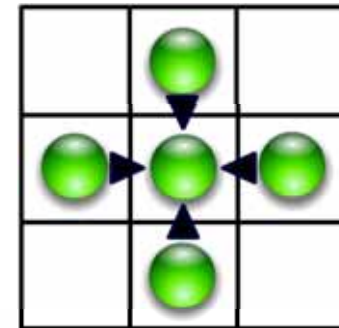
Scatter vs. Gather

- Some kernels must obtain information from cells other than one currently being processed.
- Two types of communication

- Gather
- Scatter



Scatter

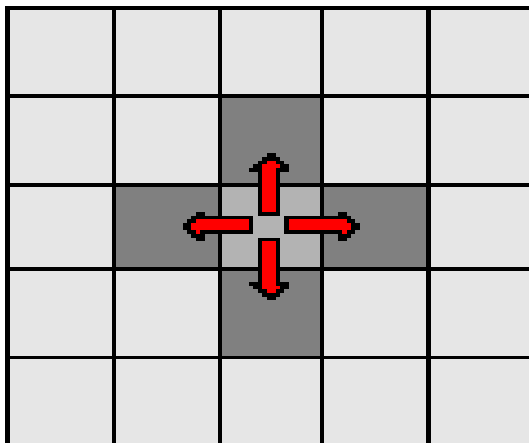


Gather

Scatter and Gather

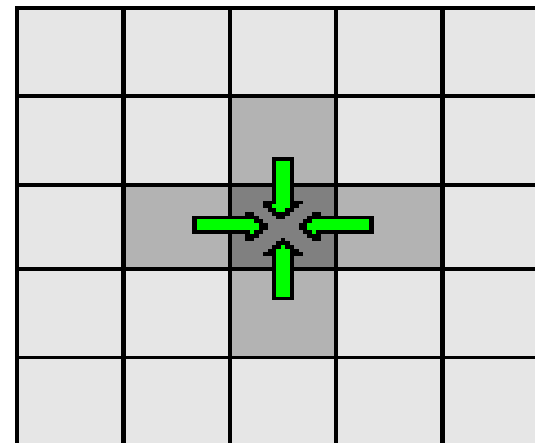
- Scatter:

- indirect write
- $a[i] = x$
- modify output address
- not possible



- Gather:

- indirect read
- $x = a[i]$
- fetch texture data
- easy



Typical GPU Computation

- Initialize “view” (so that pixels:texels::1:1)
 - `glMatrixMode(GL_MODELVIEW);`
 - `glLoadIdentity();`
 - `glMatrixMode(GL_PROJECTION);`
 - `glLoadIdentity();`
 - `glOrtho(0, 1, 0, 1, 0, 1);`
 - `glViewport(0, 0, outTexResX, outTexResY);`
- For each algorithm step:
 - Activate render-to-texture
 - Setup input textures, fragment program
 - Draw a full-screen quad (1x1)

The Nest logo is a stylized, light gray graphic of a bird's nest with three eggs, positioned behind the text 'Nest'.

Array addition using Graphics API

C++ code running in CPU

```
for( int i = 0; i < ARRAY_SIZE * ARRAY_SIZE; i++)
{
    c[i] = a[i] + b[i];
}
```

HLSL Code running in GPU

```
Texture Tex1
Texture Tex2
Sampler2D Sampler1 = Sampler_state
{
    Texture = (Tex2);
    MinFilter = Point;
    MagFilter = Point;
}
Sampler2D Sampler2 = sampler_state
{
    Texture = (Tex1);
    MinFilter = Point;
    MagFilter = Point;
};
struct V2P
{
    float4 Position : POSITION;
    float4 Tex0: TEXCOORD0;
}
float4 PS_ArrMul(V2P v) : COLOR0
{
    float4 fIResult = float4(0.0, 0.0, 0.0, 0.0);
    fIResult.r = tex2D(Sampler1, v.Tex0.xy).r +
    tex2D(Sampler2, v.Tex0.xy).r;
    return fIResult;
}
```

The Nest logo is a stylized, lowercase 'nest' in a grey, sans-serif font. It is positioned in the bottom left corner of the slide, partially overlapping a large, faint, light-grey graphic of a soccer ball.

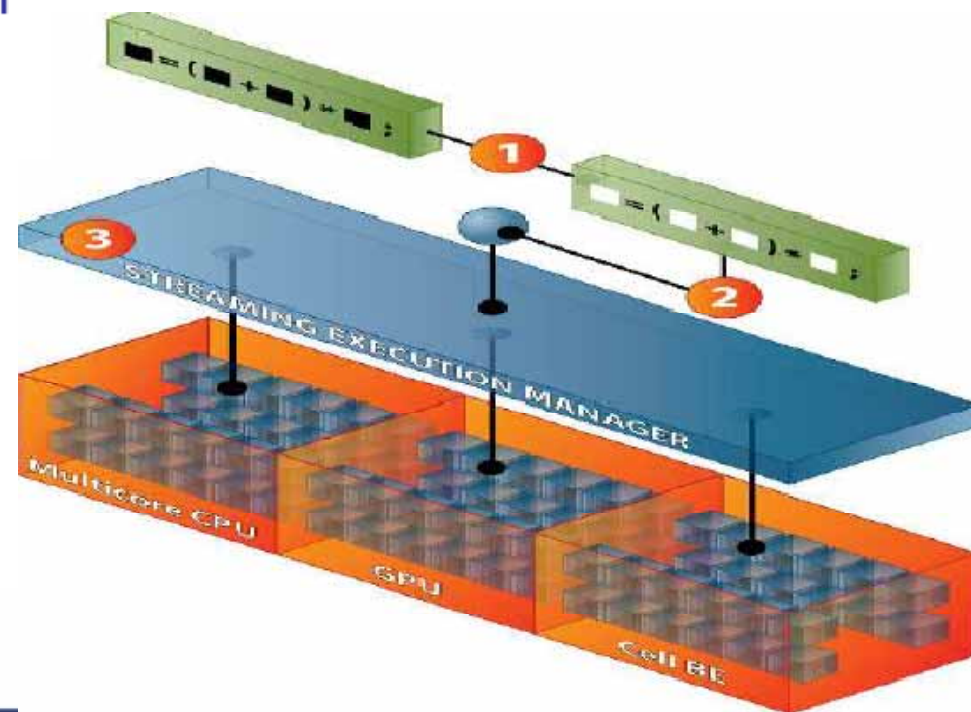
Development Platforms

- No Graphics knowledge for programming
- Knowledge of C++/C is sufficient
- Currently two different platforms
 - **Rapid Mind** (www.rapidmind.net)
 - Available for multi-core processors and GPUs.
 - Scalable to arbitrary number of cores
 - **CUDA** (developer.nvidia.com/cuda)
 - Can only work in NVIDIA architectures
 - Gives better result than other tools

RapidMind

How to convert C++ Applications to RapidMind

1. Replace Types
2. Capture Computation
3. Parallel Execution



NeST

Programming in RapidMind

C++

```
#include <cmath>

float f;
float a[512][512][3];
float b[512][512][3];

for (int x = 0; x<512; x++) {
    for (int y = 0; y<512; y++) {
        for (int k = 0; k<3; k++) {
            a[y][x][k] = f *
                (a[y][x][k] + b[y][x][k]);
        }
    }
}
```

RapidMind

```
#include <rapidmind/platform.hpp>
using namespace rapidmind;

Value1f f;
Array<2, Value3f> a(512, 512);
Array<2, Value3f> b(512, 512);

Program prog = BEGIN {
    In<Value3f> r, s;
    Out<Value3f> q;
    q = f * (r + s);
} END;
```

A Word of Caution

- **Limitations of Stream Programming**
 - Not all applications are amenable to stream programming
 - Data streams and processing kernels are not very obvious always
 - Lack of error handling
 - Lack of debugging support
- **Limitations of GPGPU**
 - Need to optimize algorithms to leverage hardware strengths
 - Accommodating large data sizes, due to video memory size limitation
 - Cost of data transfer from the specialized cores to CPU
 - Accuracy of floating point operations, although this is improving
 - Changing features with every model/variant of GPU

There is no silver bullet, of course!

Thank you

Any Questions

NeST