NVIDIA®

# GPU Computing

## Programming a Massively Parallel Processor

## CGO-5

Ian Buck

Stunning Graphics Realism

Lush, Rich Worlds

Crysis © 2006 Crytek / Electronic Arts

POWERED BY NVIDIA

Incredible Physics Effects

Core of the Definitive Gaming Platform

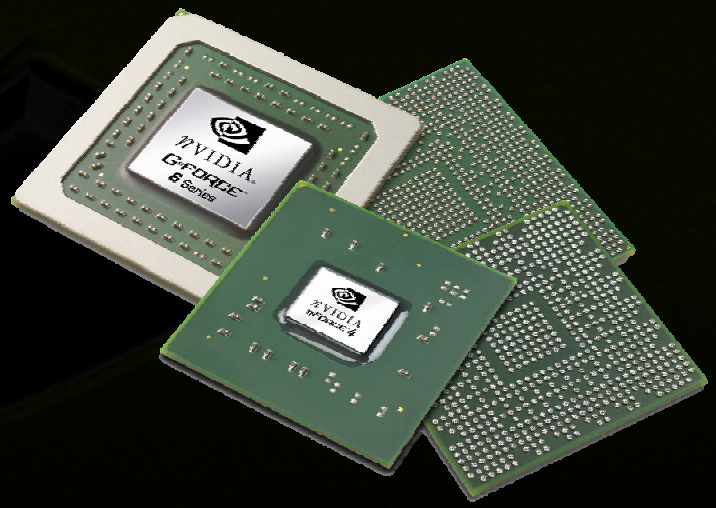
Hellgate: London © 2005-2006 Flagship Studios, Inc. Licensed by NAMCO BANDAI Games America, Inc.

Full Spectrum Warrior: Ten Hammers © 2006 Pandemic Studios, LLC. All rights reserved. © 2006 THQ Inc. All rights reserved.

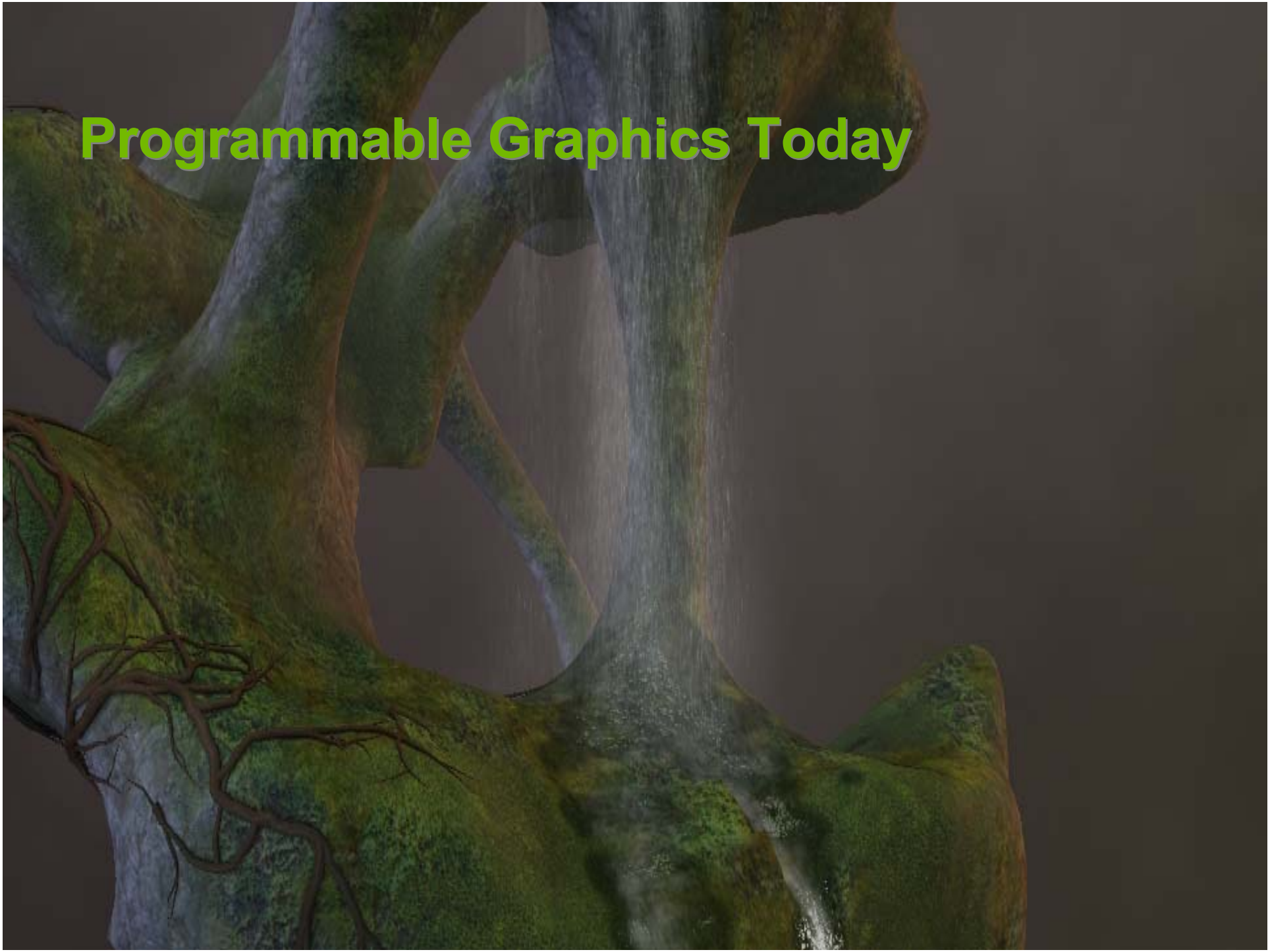
# GPUs: Graphics Processing Units

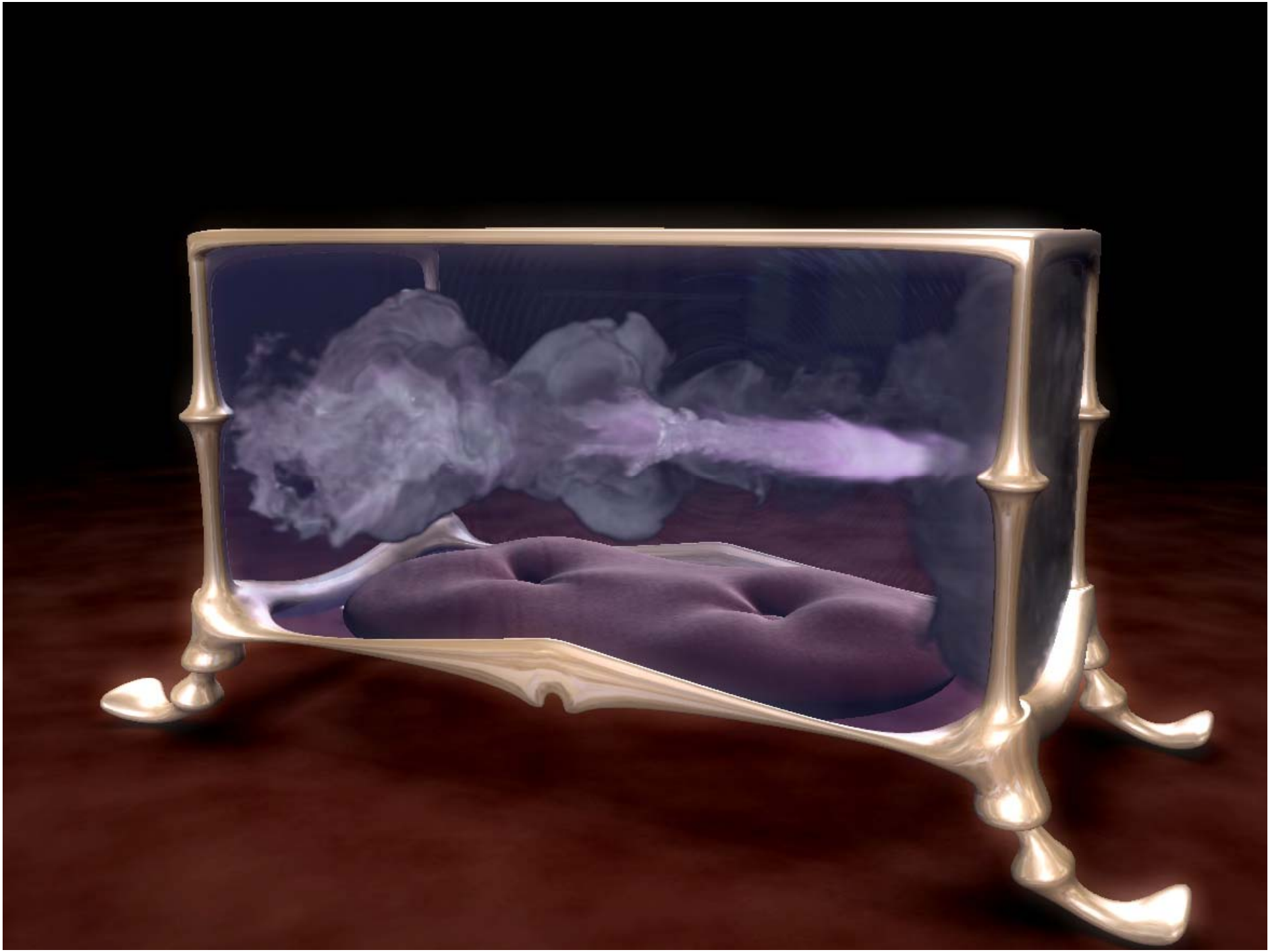

- Programmable Units also called traditionally as shader units. (e.g. Vertex processor, Pixel processor)

- As with any programmable processor, these need to be programmed and compiled efficiently for performance.



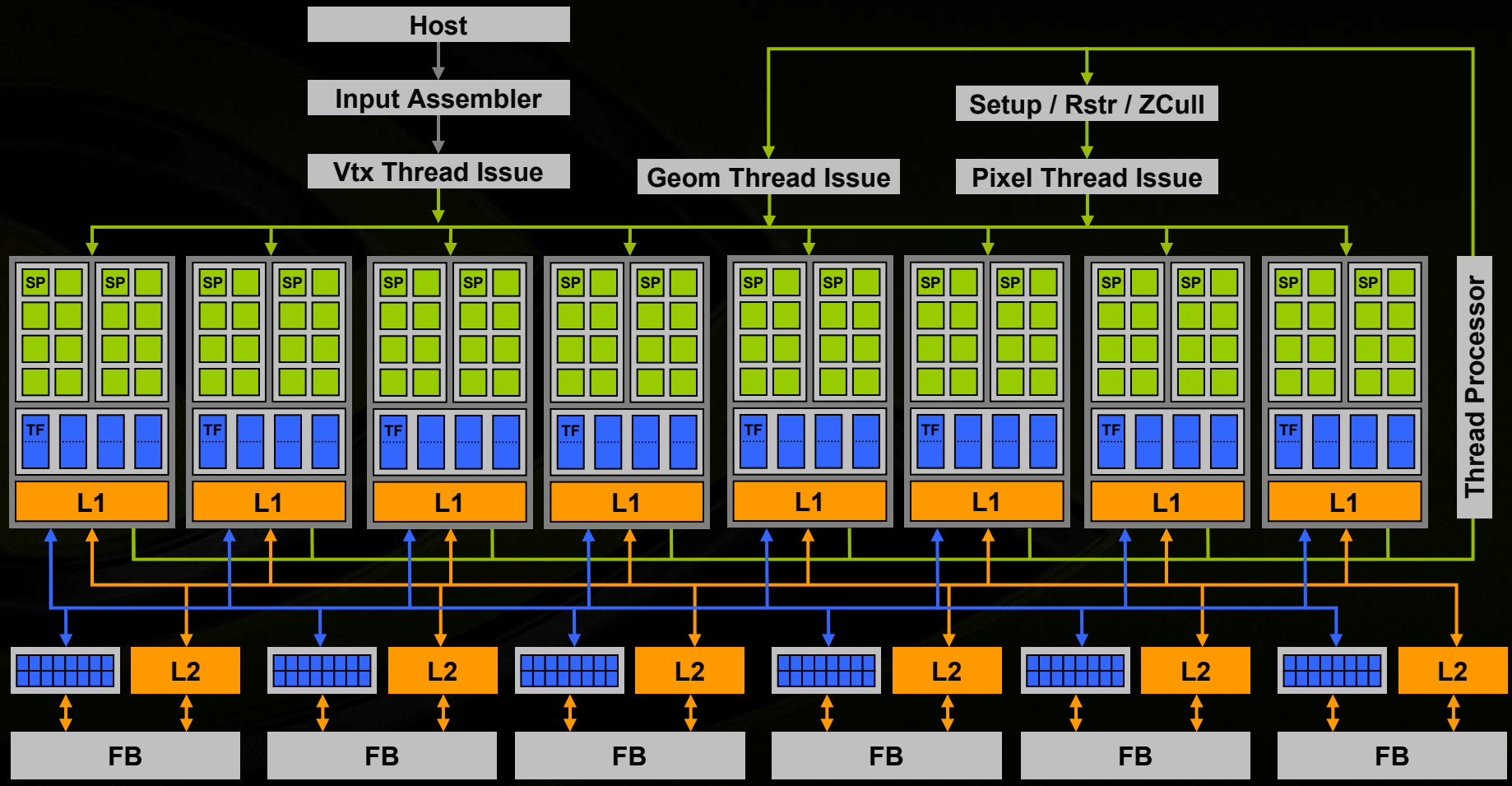
# Programmable Graphics Today





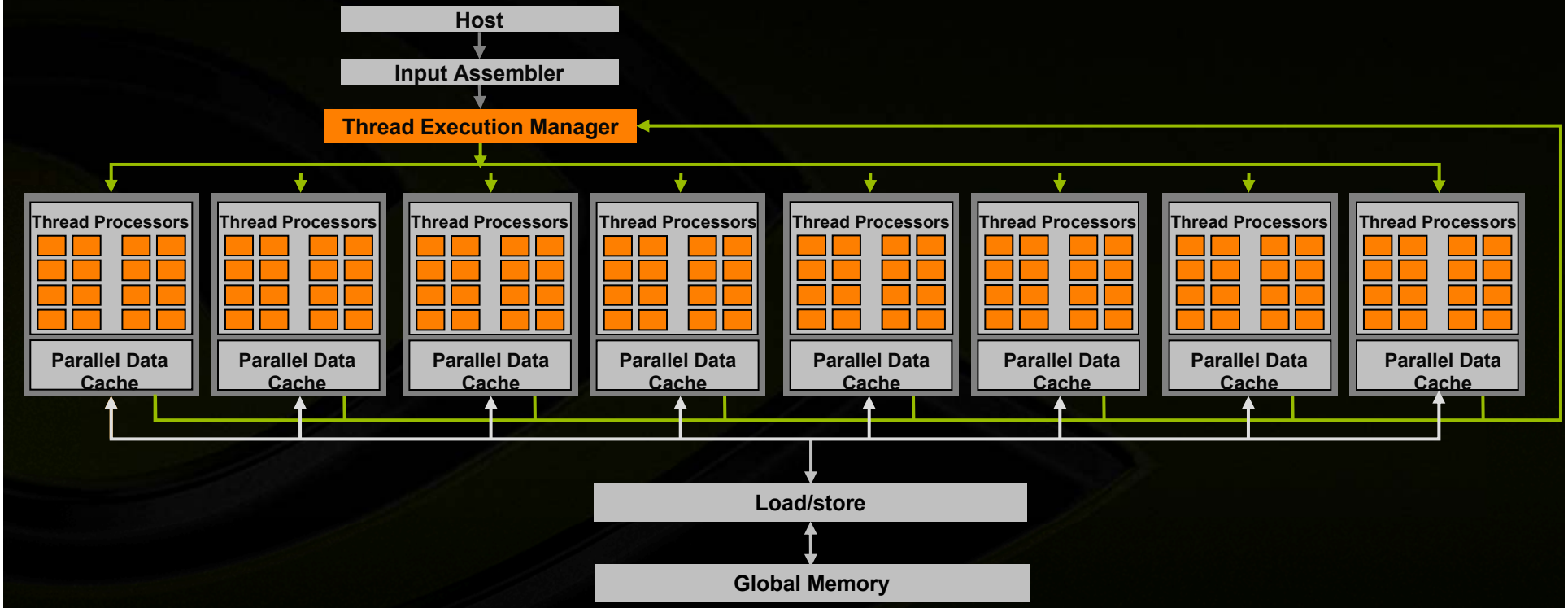
# GeForce 8800

- Build the architecture around the processor



# GeForce 8800 GPU Computing

- Next step: Expose the GPU as massively parallel processors



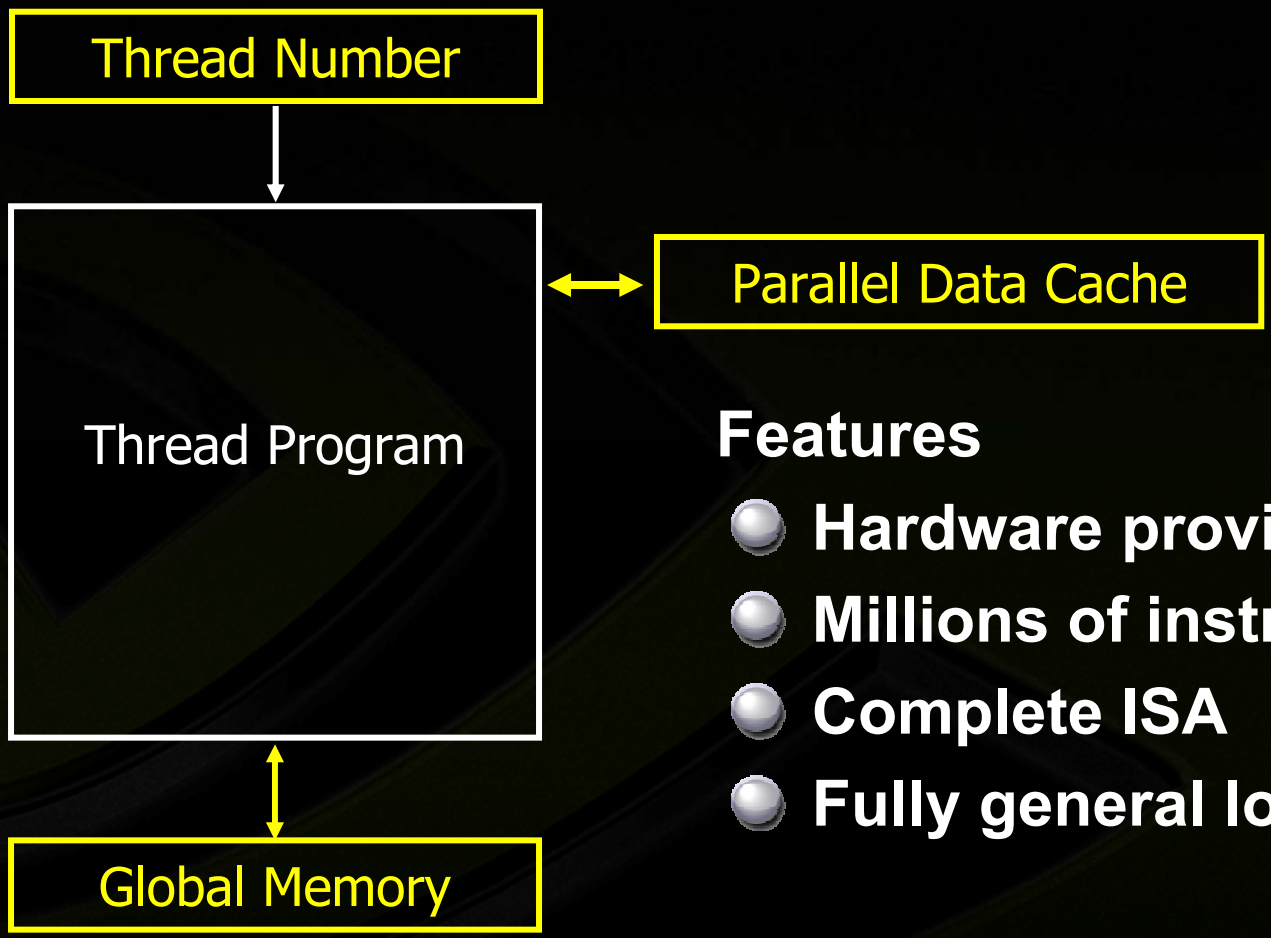
# Outline


- CUDA & GPU Computing
- Compiling CUDA
- NVIDIA compiler challenges
- Research opportunities

NVIDIA®

# GPU Computing Model

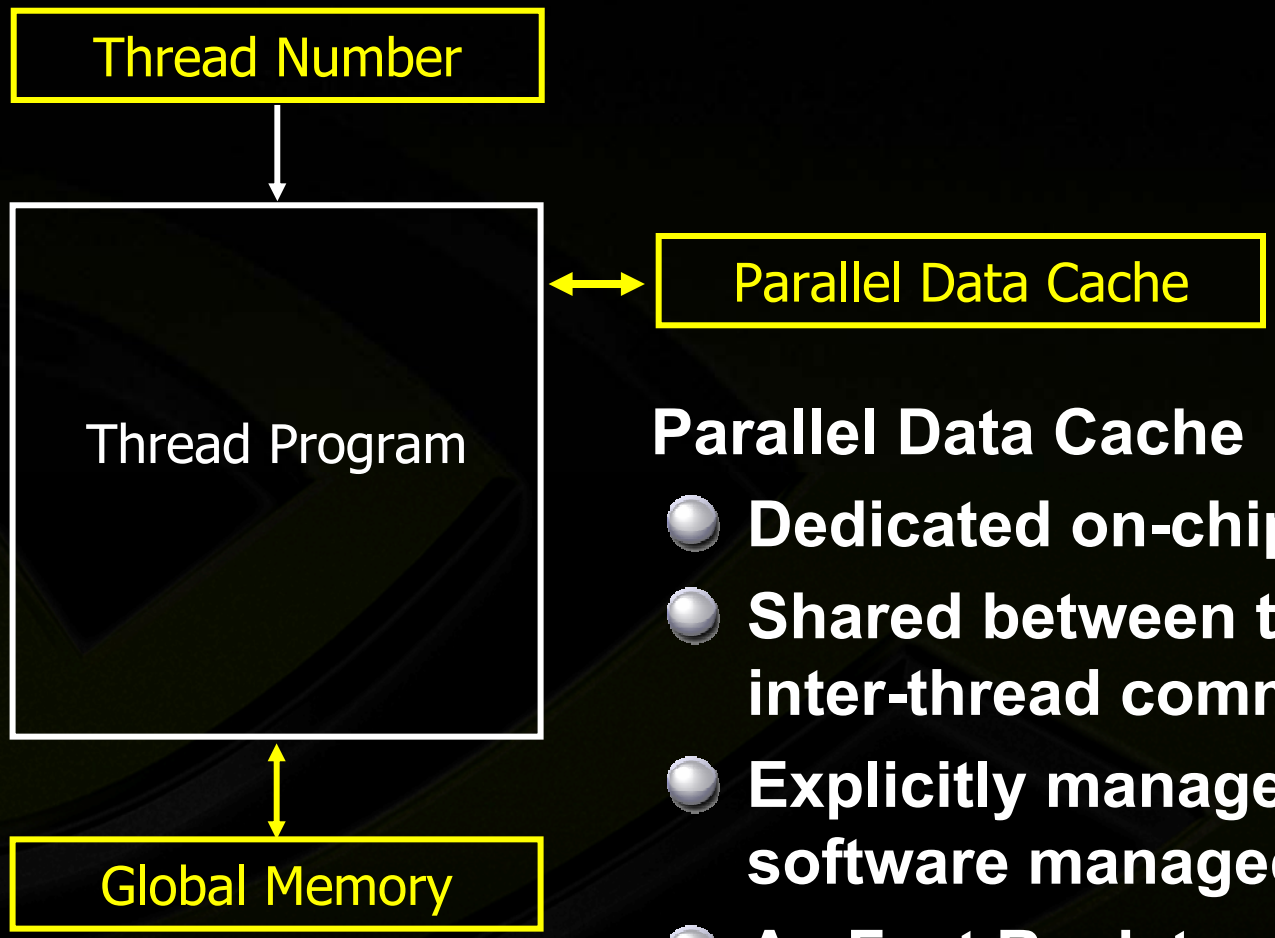
# Thread Programs




**Features**

- **Hardware provided thread id**
- **Millions of instructions**
- **Complete ISA**
- **Fully general load/store**

# Thread Programs




**Parallel Data Cache**

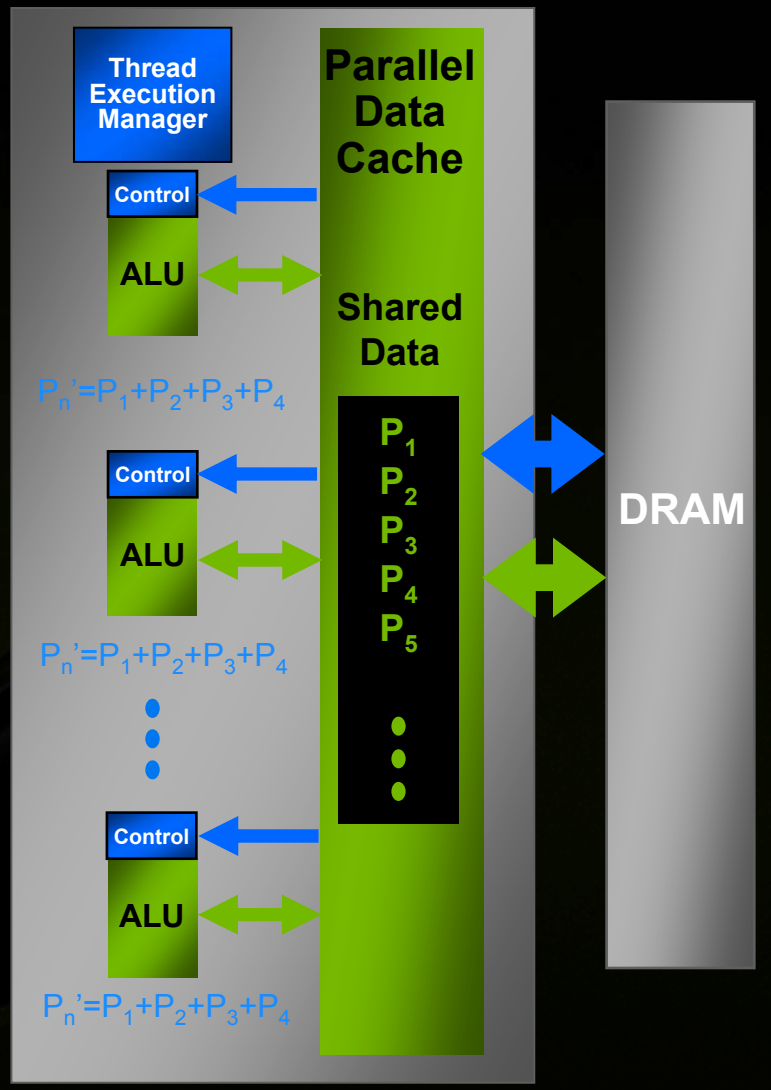
- **Dedicated on-chip memory**
- **Shared between threads for inter-thread communication**
- **Explicitly managed – software managed cache**
- **As Fast Registers**

# Parallel Data Cache

## Bring the data closer to the ALU

- Stage computation for the parallel data cache
- Minimize trips to external memory
- Share values to minimize overfetch and computation
- Increases arithmetic intensity by keeping data close to the processors

**Addresses a fundamental problem of stream computing**



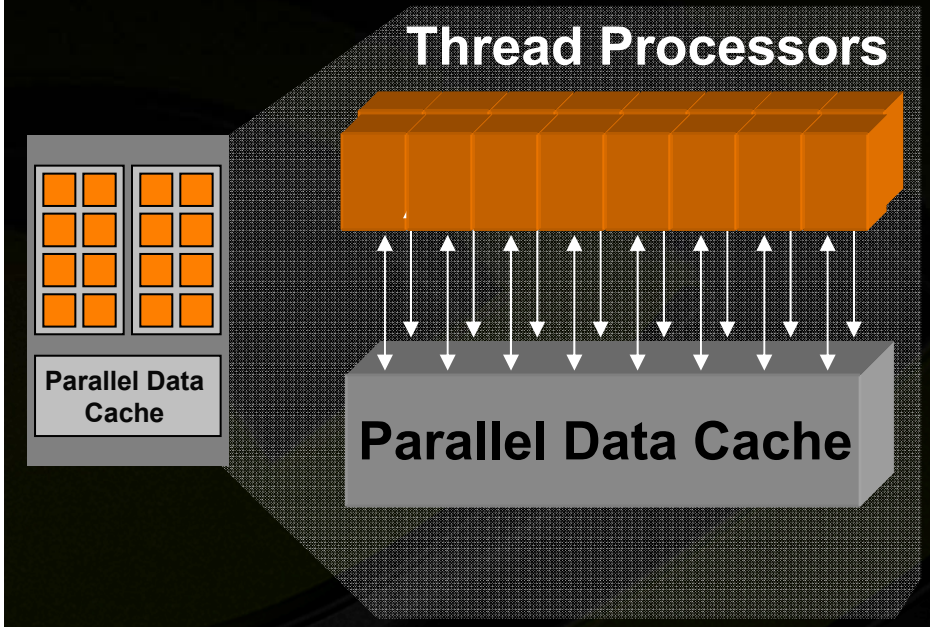

Parallel execution through cache

# GeForce 8800 GTX Graphics Board


**$599 e-tail**

| | |
|---|---|
| Core | 575MHz |
| Multi-Processors | 128 |
| Shader | 1350MHz |
| Memory | 900MHz |
| Memory | 768MB GDDR3 |

# Thread Processor




- 128, 1.35 GHz processors
- 16KB Parallel Data Cache per cluster
- Scalar architecture
- IEEE 754 Precision
- Full featured instruction set

# A Massively Parallel Coprocessor


- The GPU is viewed as a compute device that:
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs 1000's of threads in parallel
- Data-parallel portions of an application execute on the device as kernels which run many cooperative threads in parallel
- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - Very little creation overhead
  - GPU needs 1000s of threads for full efficiency
    - Multi-core CPU needs only a few
  - Threads are non-persistent
    - Run and exit

# CPU / GPU Parallelism


- Moore's Law gives you more and more transistors
  - What do want to do with them?
- CPU Strategy: Make the workload (one compute thread) run as fast as possible
  - Tactics
    - Cache (area limiting
    - Instruction/Data Prefetch
    - "hyperthreading"
    - Speculative Execution
    - → limited by "perimeter" – communication bandwidth
    - ...then add task parallelism... Multi-core
- GPU Strategy: Make the workload (as many threads as possible) run as fast as possible
  - Tactics
    - Parallelism (1000s of threads)
    - Pipelining
    - → limited by "area" – compute capability

NVIDIA®

# GPU Computing: Compiling

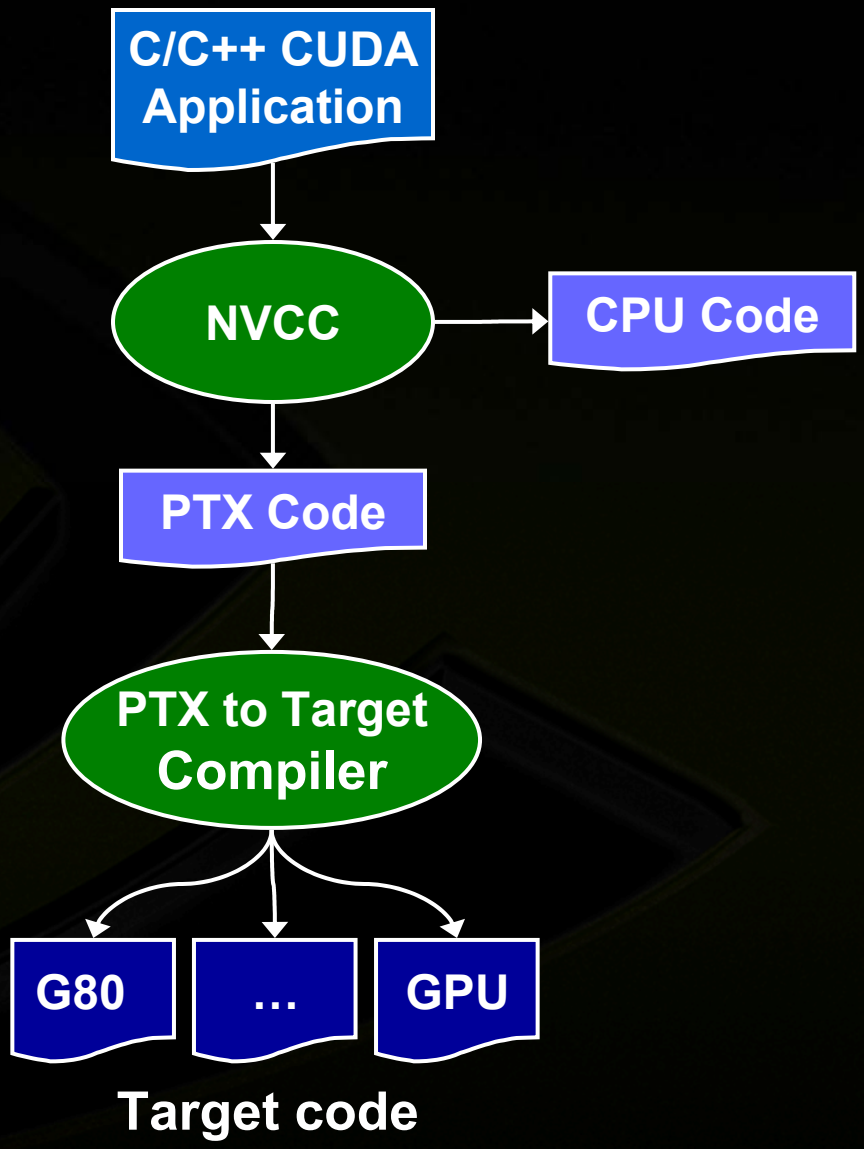
# CUDA: C on the GPU


- A simple, explicit programming language solution
- Extend only where necessary

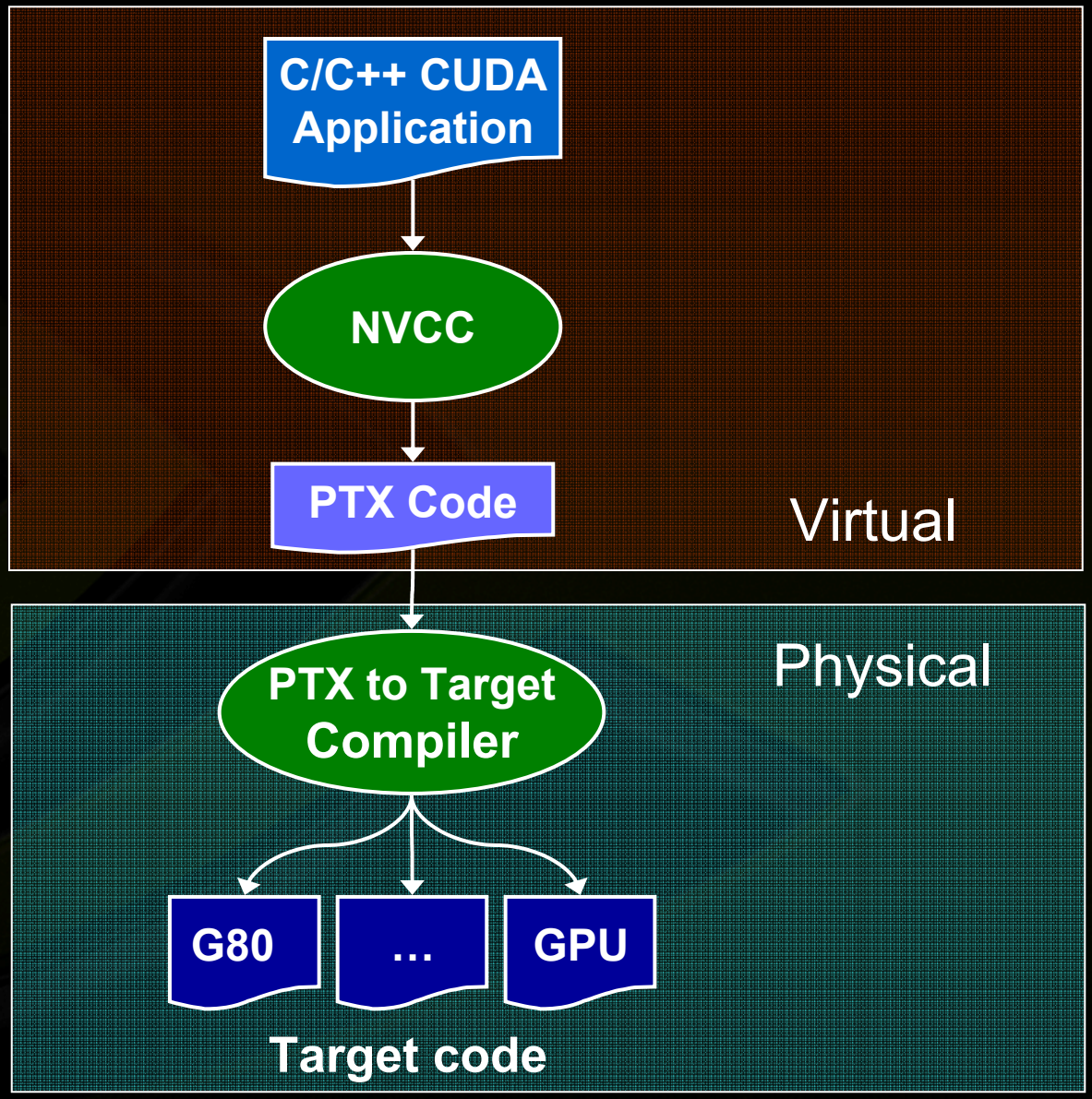
```
__global__ void KernelFunc(...);

__shared__ int SharedVar;


KernelFunc<<< 500, 128 >>>(...);
```

- Explicit GPU memory allocation
  - `cudaMalloc()`, `cudaFree()`
- Memory copy from host to device, etc.
  - `cudaMemcpy()`, `cudaMemcpy2D()`, ...

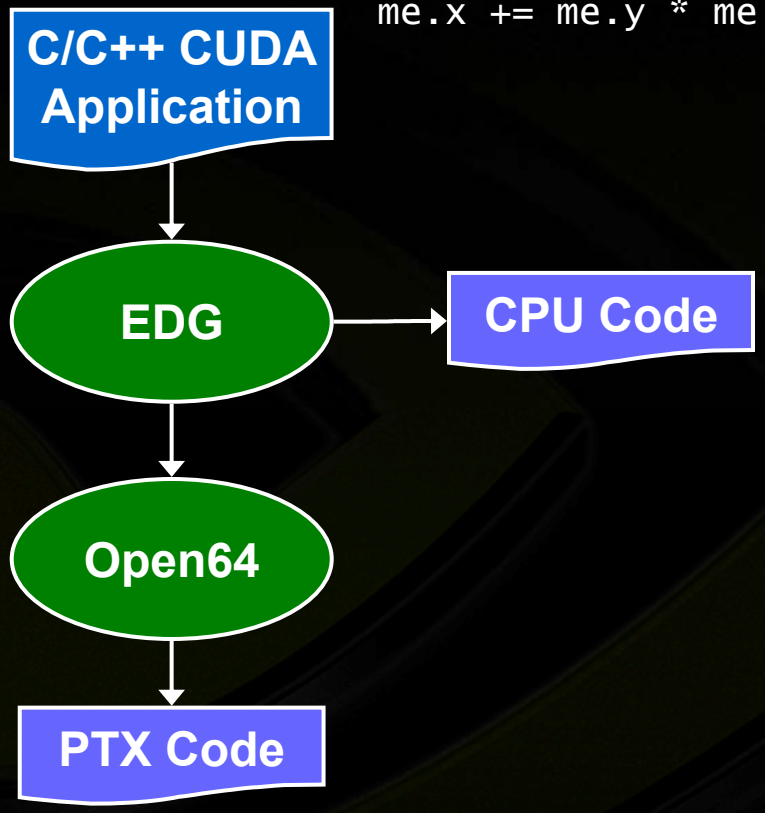
# Compiling CUDA



# Compiling CUDA



# NVCC & PTX Virtual Machine




```
float4 me = gx[gtid];
me.x += me.y * me.z;
```

- EDG
  - Separate GPU vs. CPU code
- Open64
  - Generates GPU PTX assembly
- Parallel Thread eXecution (PTX)
  - Virtual Machine and ISA
  - Programming model
  - Execution resources and state

```
ld.global.v4.f32  {$f1,$f3,$f5,$f7}, [$r9+0];
mad.f32           $f1, $f5, $f3, $f1;
```

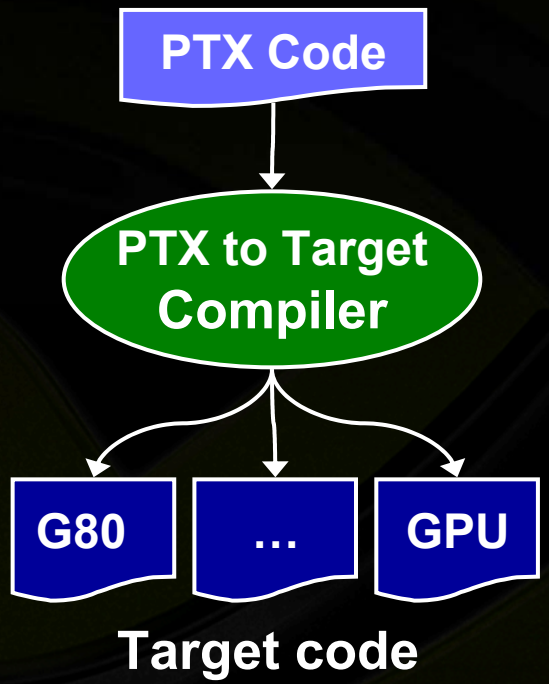
# Role of Open64


Open64 compiler gives us

- A complete C/C++ compiler framework. More futuristic looking. We do not need to add infrastructure framework as our hardware arch advances over time.
- A good collection of high level architecture independent optimizations. All GPU code is in the inner loop.
- Compiler infrastructure that interacts well with other standardized related tools.

# Virtual to Physical ISA translation

```
ld.global.v4.f32  {$f1,$f3,$f5,$f7}, [$r9+0];
mad.f32           $f1, $f5, $f3, $f1;
```




```
0x103c8009 0x0fffffff
0xd00e0609 0xa0c00780
0x100c8009 0x00000003
0x21000409 0x07800780
```

- NVIDIA developed GPU backend compiler
  - Leveraged by all NVIDIA products
- Target specific optimizing
  - ISA diffences
  - Resource allocation
  - Performance

# Challenges for PTX compilation


- Runtime code generation has
  - stringent compile-time and memory constraints
  - but since each program is a innermost kernel (think of 1600X1200 doubly nested loop outside each program), performance optimizations are critical.
  - Allow pre-compiled target GPU binaries.
- Criteria that affect performance can be different from traditional compilers. So, some of the problems are not addressed well in literature.
- Architecture changes and capability changes occur frequently, so compiler framework has to be flexible in addition to being optimal and efficient.

# Challenge: Resource Pressure = Performance


- Using more registers hinders parallelism, using spills is extremely slow. Need to use minimal number of registers.
- Some architectures were VLIW and required lot of ILP (instruction level parallelism) for perf. Combined with constraints on registers, the problem is extremely challenging.
- Batching memory access for performance reasons. Again, combined with constraints of registers, this problem is very challenging.

# Challenges Continued


- Competing optimizers
  - Open64 lifting code from loops increases register pressure.
- Synchronization across control flow within programs.
- Phase ordering of transformations and optimizations is a continual concern.

# CUDA Performance Advantages


**Performance:**

- BLAS1: 60+ GB/sec
- BLAS3: 100+ GFLOPS
- FFT: 52 benchFFT* GFLOPS
- FDTD: 1.2 Gcells/sec
- SSEARCH: 5.2 Gcells/sec
- Black Scholes: 4.7 GOptions/sec

**How:**

- Leveraging the parallel data cache
- GPU memory bandwidth
- GPU GFLOPS performance
- Custom hardware intrinsics
  - __sinf, __cosf, __expf, __logf, ...

**All benchmarks are compiled code!**

*GFLOPS as defined by http://www.fftw.org/benchfft

# Research Opportunities


- GPU Computing a new architecture with new opportunities for compiler research and innovation
- What NVIDIA provides
  - Explicit C/C++ compilation tool chain
  - PTX assembly specification
  - PTX enabled Open64 sources
  - $599 supercomputer at Fry's

# Research Opportunities


- Considering variable-sized resources in optimizers
  - Registers
  - Shared memory
  - Threads
- Data parallel languages
  - Functional
  - Array Based
  - Domain Specific
- Blocking in massively parallel applications
  - Compiler control over parallel data cache for blocking
  - Discovering thread block size
- GPU Onload
  - Discovering portions of CPU code to “onload” to GPU

# Research Opportunities


- Rethinking recompilation
  - Cheaper to recompute than save-restore
  - “FLOPS are Free” computing
- Compiling for the memory hierarchy
  - “Bandwidth is expensive”
- Education
  - Where are the worlds parallel programmers?

# Future


- GPUs are already at where CPU are going
  - CPU today = 8 cores
  - GeForce 8800 =  128 cores
- Task parallelism is short lived...
- Data parallel is the future
  - Express a problem as data parallel....
  - Maps automatically to a scalable architecture
- CUDA provides an insight into a data parallel future