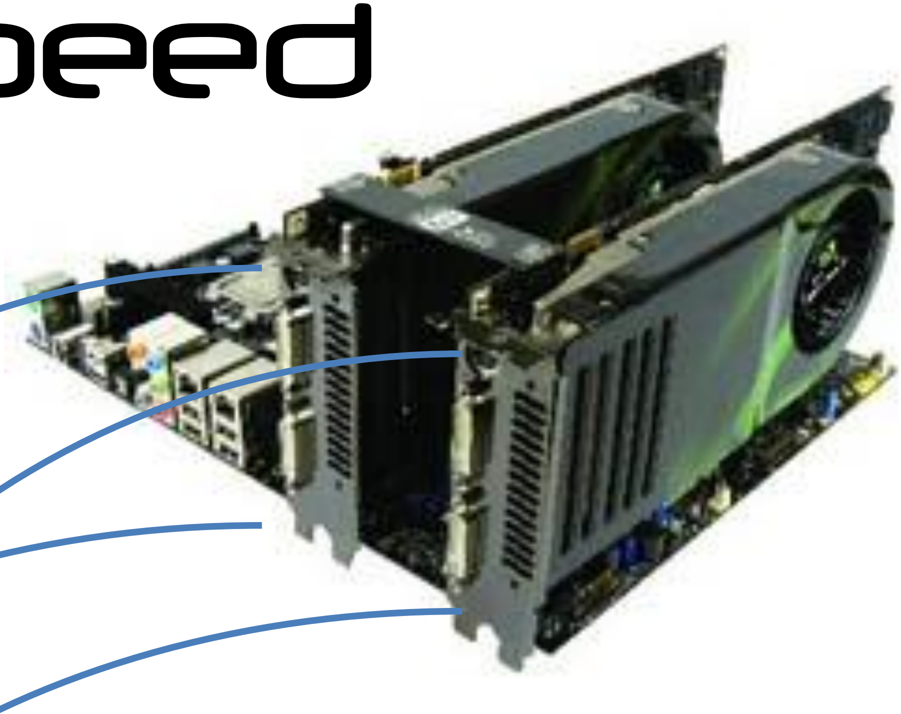
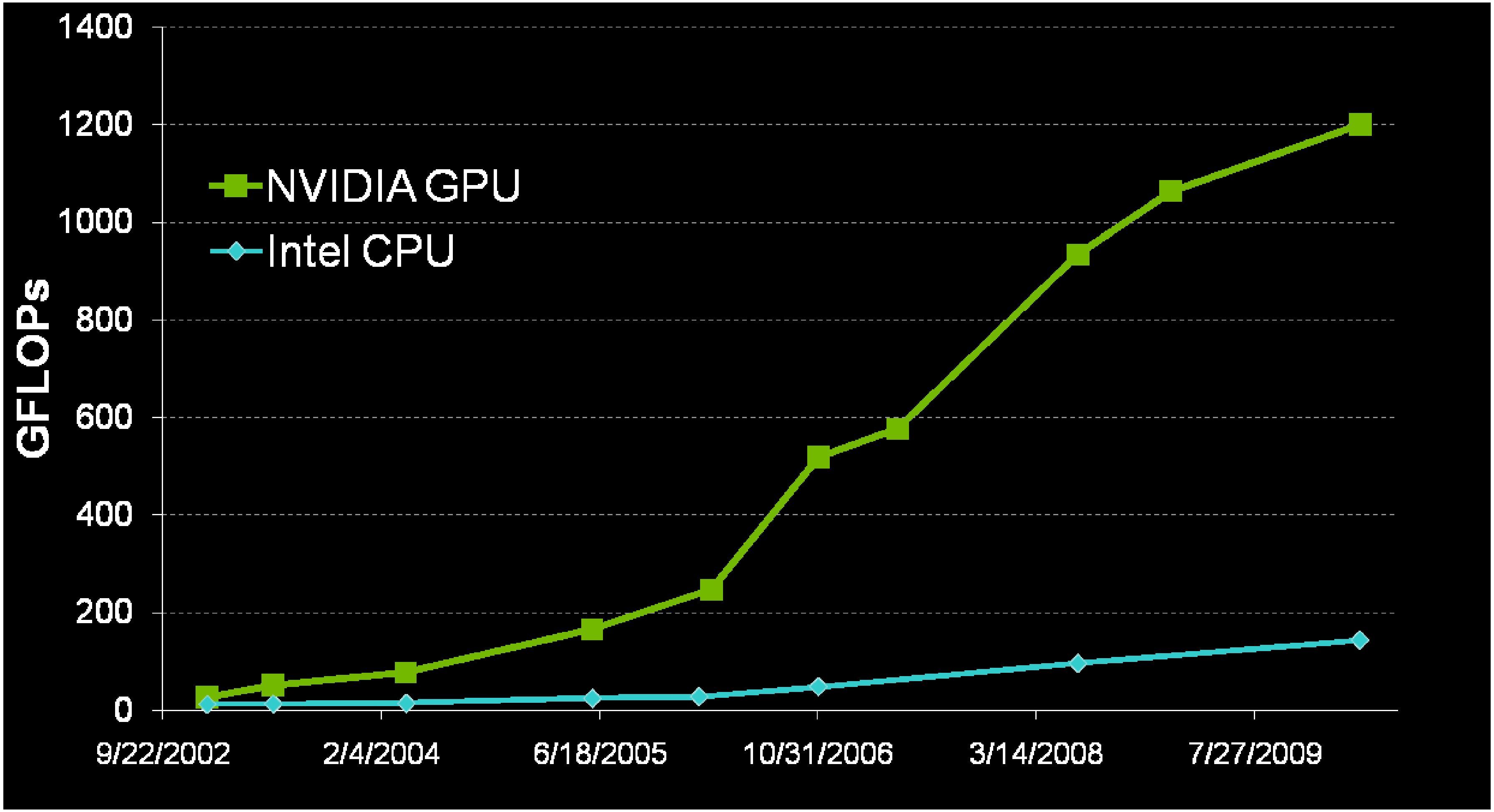
Warp Speed

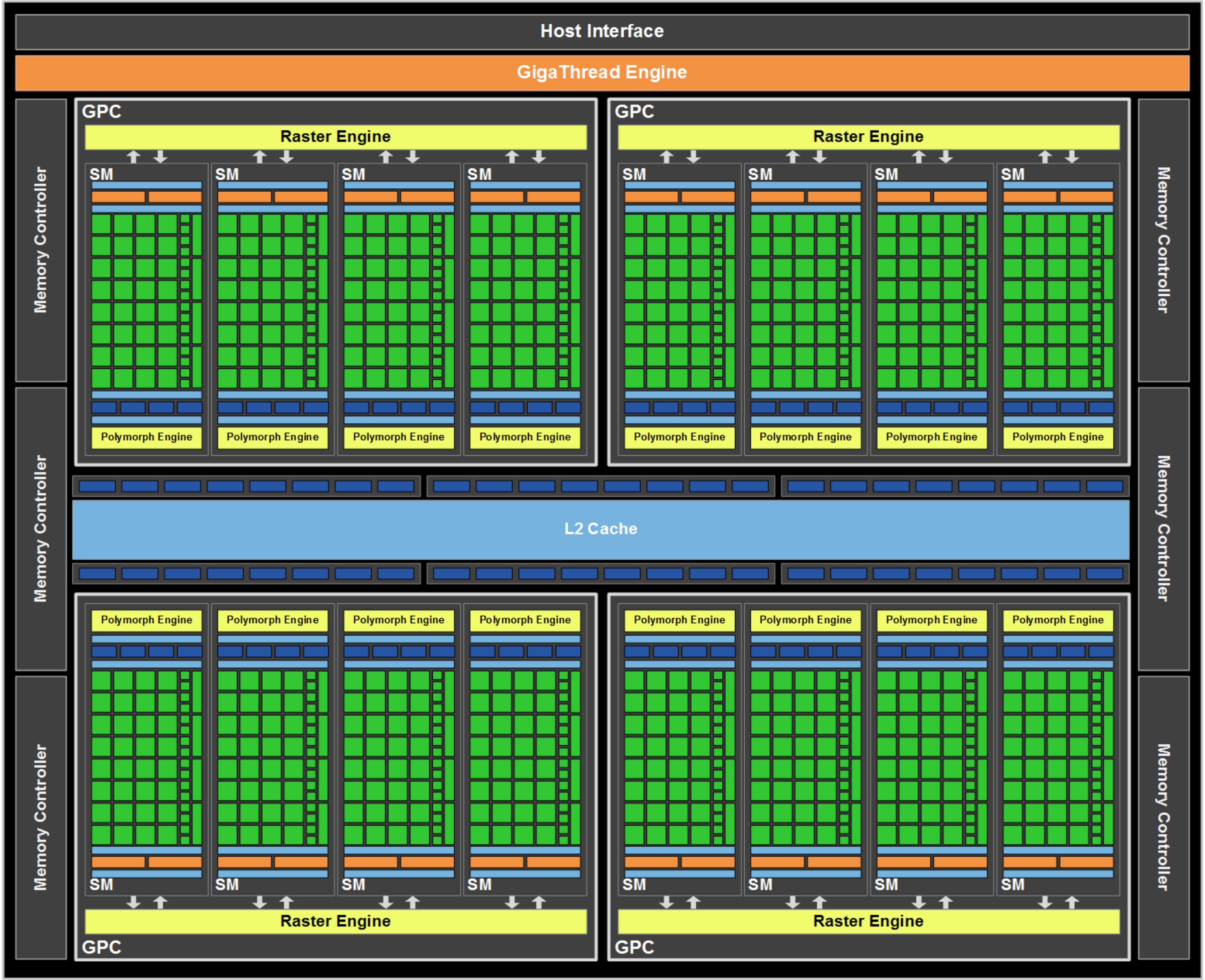


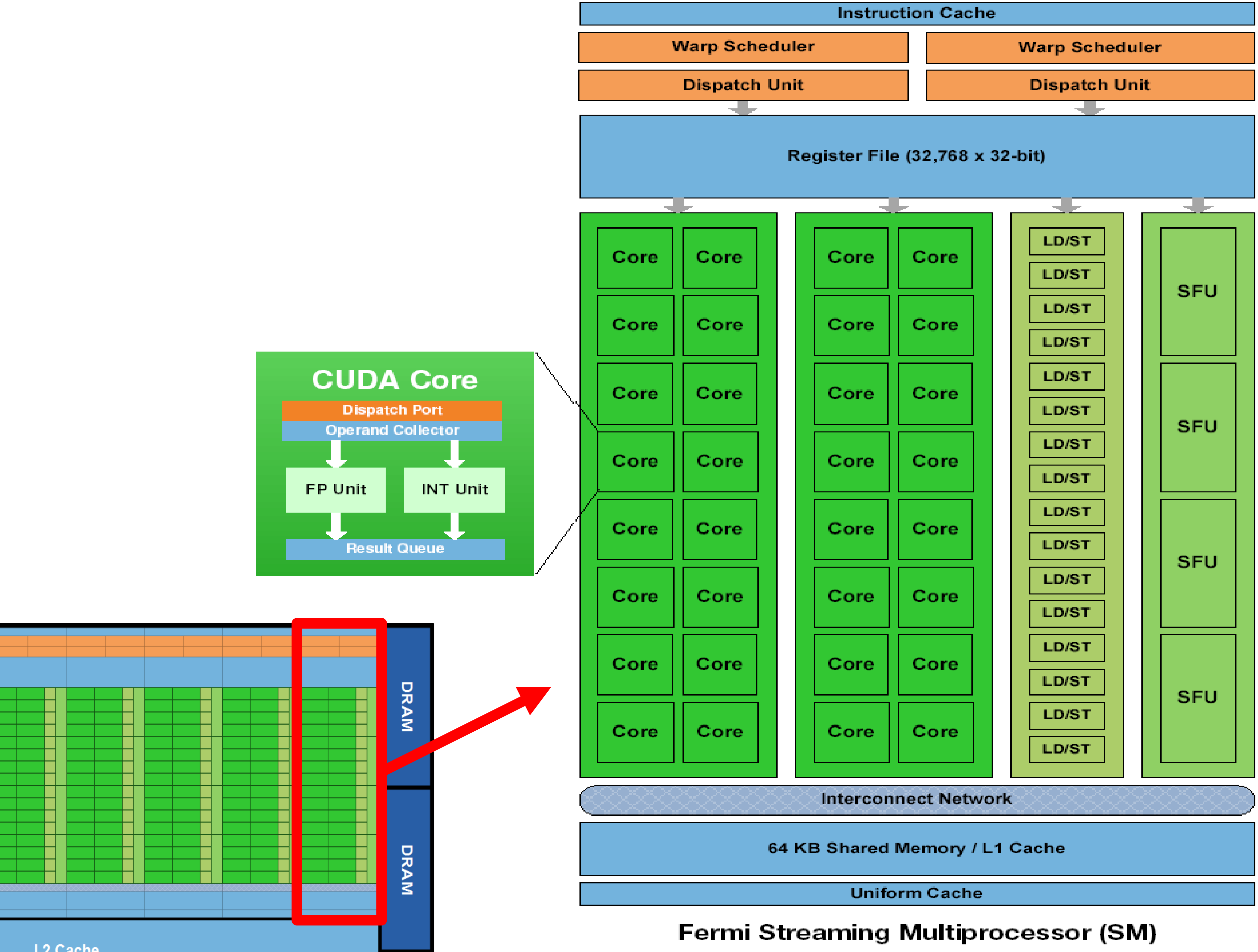
A lighthearted introduction to GPGPUs

Dr. Keith Schubert

Why Bother?


1400
1200
1000
800
600
400
200
0
GFLOPs
NVIDIA GPU
Intel CPU
9/22/2002
2/4/2004
6/18/2005
10/31/2006
3/14/2008
7/27/2009


Host Interface
GigaThread Engine
GPC
Raster Engine
SM
Polymorph Engine
Memory Controller
L2 Cache



Instruction Cache

Warp Scheduler | Warp Scheduler

Dispatch Unit | Dispatch Unit

Register File (32,768 x 32-bit)

Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core

LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST LD/ST

SFU SFU SFU SFU

Interconnect Network

64 KB Shared Memory / L1 Cache

Uniform Cache

CUDA Core

Dispatch Port

Operand Collector

FP Unit | INT Unit

Result Queue

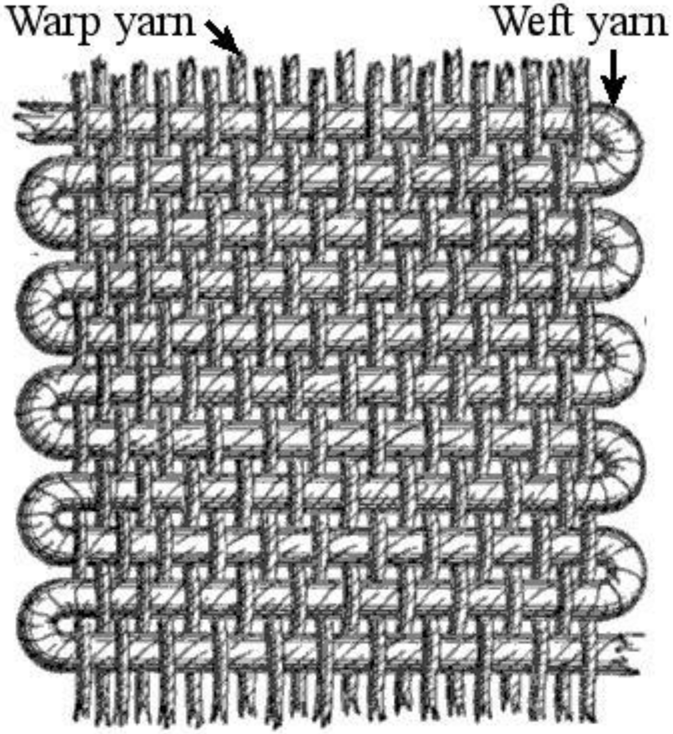
DRAM

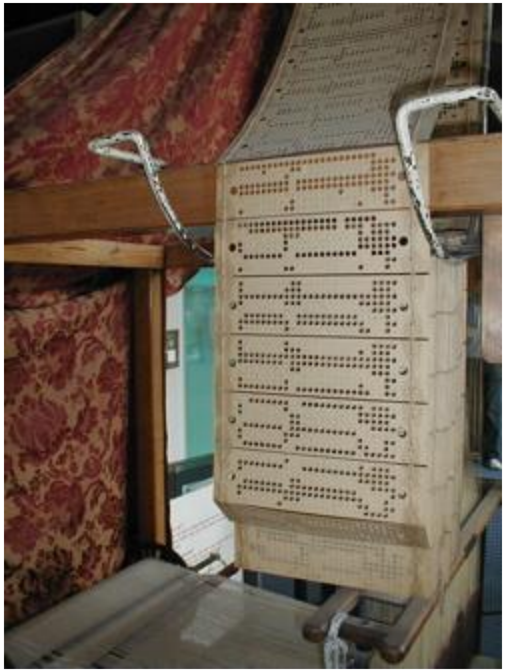
DRAM

L2 Cache

Fermi Streaming Multiprocessor (SM)

Jacquard Looms Again



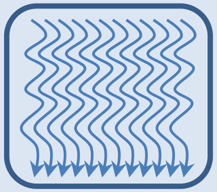


Thread

Registers (F)
type var[n];
Local Memory(S)
type var[n];

Index
threadIdx.x

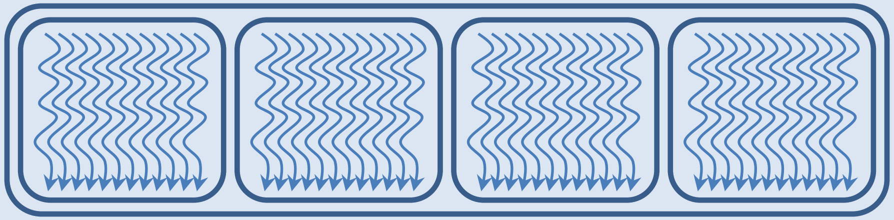
Block



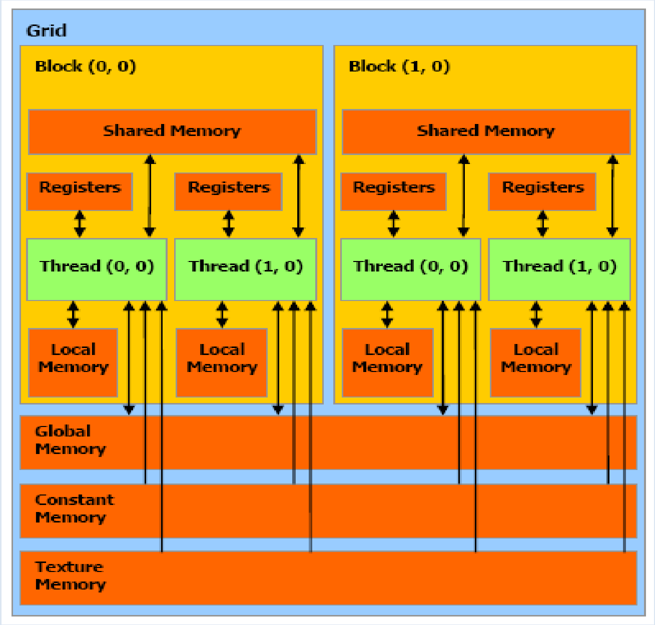
Shared Memory (F)
__shared__ type var;

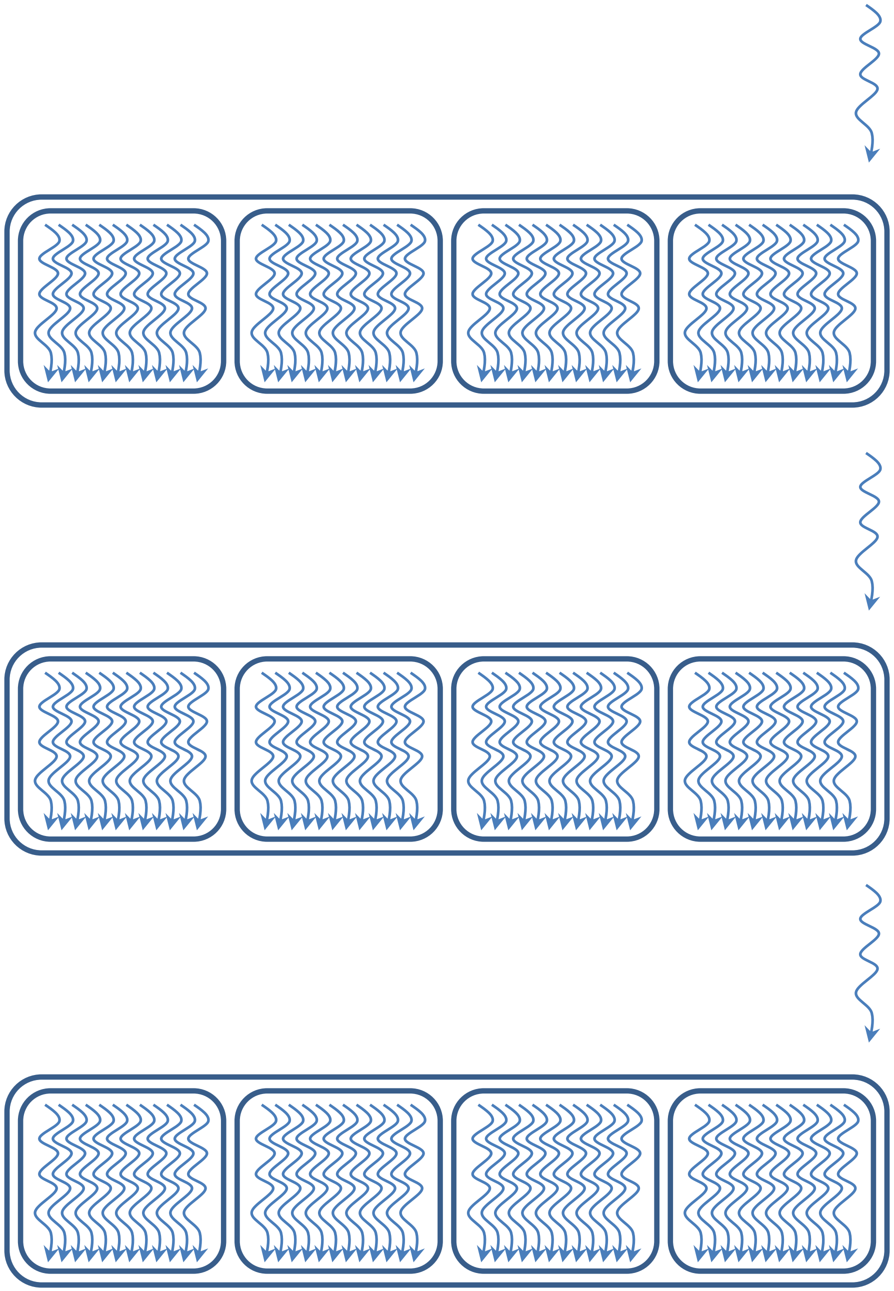
Index
blockIdx.x
Size
blockDim.x

Grid



Constant Memory (F)
__constant__ type var;
Global Memory (S)
__device__ type var;





CPU (host)

GPU (device)

CPU (host)

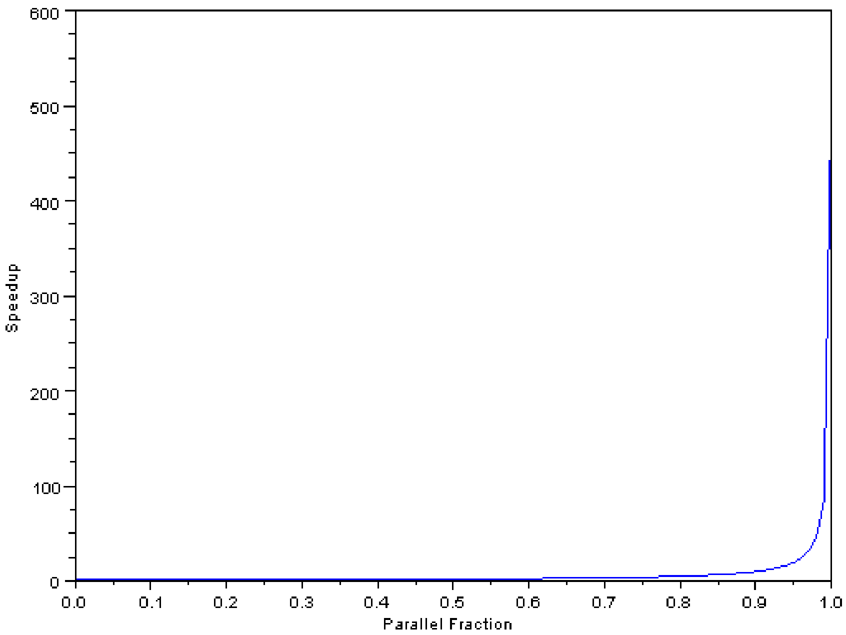
GPU (device)

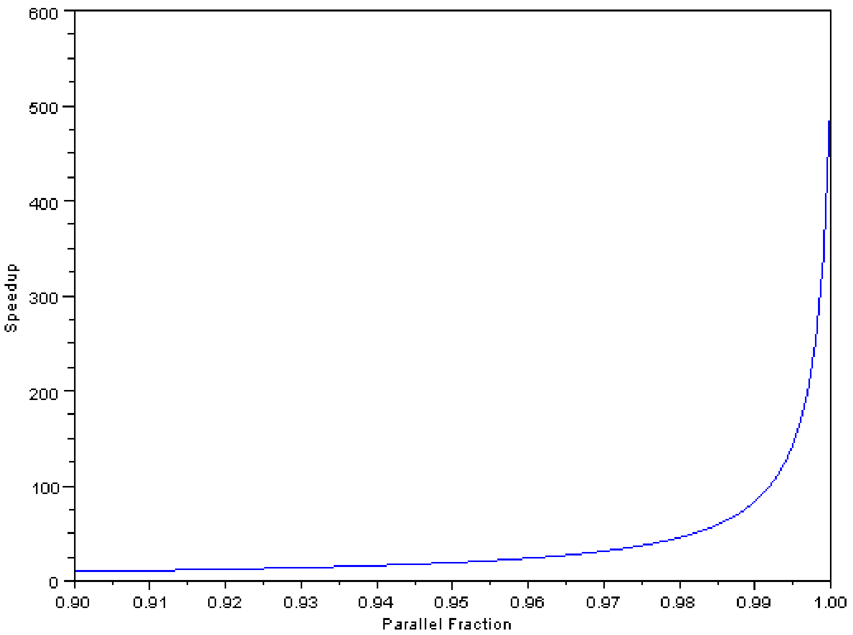
CPU (host)

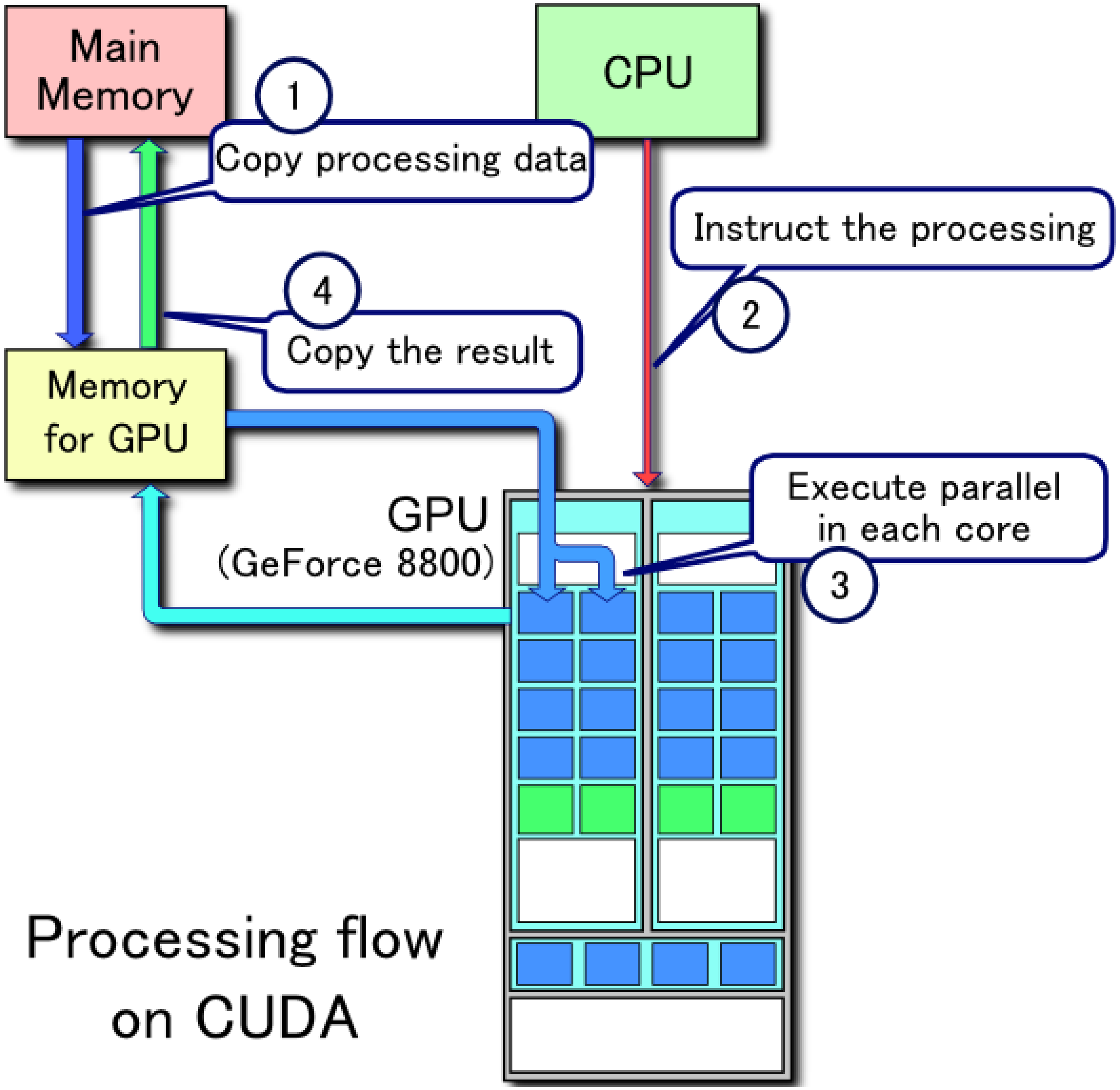
GPU (device)

Amdahl’s Law

$$Speedup = \frac{1}{f_{serial} + \frac{f_{parallel}}{512}}$$







Main Memory

CPU

1 Copy processing data

Instruct the processing 2

4 Copy the result

Memory for GPU

GPU (GeForce 8800)

Execute parallel in each core 3

Processing flow on CUDA

Vector Addition

$$\begin{bmatrix} 2 \\ 3 \\ 5 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 2 \\ 3 \end{bmatrix}$$

- Embarrassingly parallel

c[i]=a[i]+b[i]

Kernel

```
__global__ void vector_add(const float *a,
                 const float *b,
                 float *c,
                 const size_t n){
  unsigned int i = threadIdx.x + blockDim.x * blockIdx.x;
  if(i < n)
      c[i] = a[i] + b[i];
}
```

Setup Constants

```
int main(void){
  const int n_e = 1<<20;
  const int n_b = n_e * sizeof(float);
  const size_t n_tpb = 256;
  size_t n_bl = n_e / n_tpb;
  if(n_e % n_t)
              n_bl++;
```

Pointers

```
float *a_d = 0;
float *b_d = 0;
float *c_d = 0;
float *a_h = 0;
float *b_h = 0;
float *c_h = 0;

a_h = (float*)malloc(n_b);
b_h = (float*)malloc(n_b);
c_h = (float*)malloc(n_b);

cudaMalloc((void**)&a_d, n_b);
cudaMalloc((void**)&b_d, n_b);
cudaMalloc((void**)&c_d, n_b);
```

Verify and Initialize

```
if(a_h == 0 || b_h == 0 || c_h == 0 ||
   a_d == 0 || b_d == 0 || c_d == 0){
  printf("Out of memory. We wish to hold the
   whole sky, but we never will.\n");
  return 1;
 }

 for(int i = 0; i < n_e; i++){
  a_h[i] = i* ((float)rand() / RAND_MAX);
  b_h[i] = (n_e-i)* ((float)rand() / RAND_MAX);
 }
```

Using the GPU

```
cudaMemcpy(a_d, a_h, n_b, cudaMemcpyHostToDevice);
cudaMemcpy(b_d, b_h, n_b, cudaMemcpyHostToDevice);

vector_add<<<n_bl, n_tpb>>>(a_d, b_d, c_d, n_e);

cudaMemcpy(c_h, c_d, n_b, cudaMemcpyDeviceToHost);
```

Cleanup

```
  for(int i = 0; i < 20; i++)
      printf("[%d] %7.1f + %7.1f = %7.1f\n", i, a_h[i], b_h[i], c_h[i]);

  free(a_h);
  free(b_h);
  free(c_h);

  cudaFree(a_d);
  cudaFree(b_d);
  cudaFree(c_d);
}
```