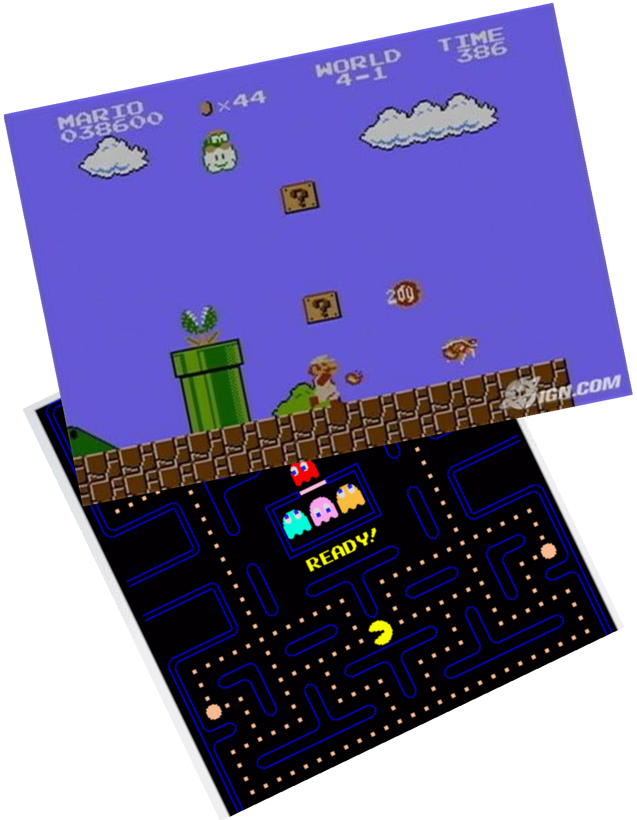
Beyond games: Supercomputer on your desktop

Meng Wang
Boston University

Kick off meeting

Games advanced so much..



Yesterday

NOW

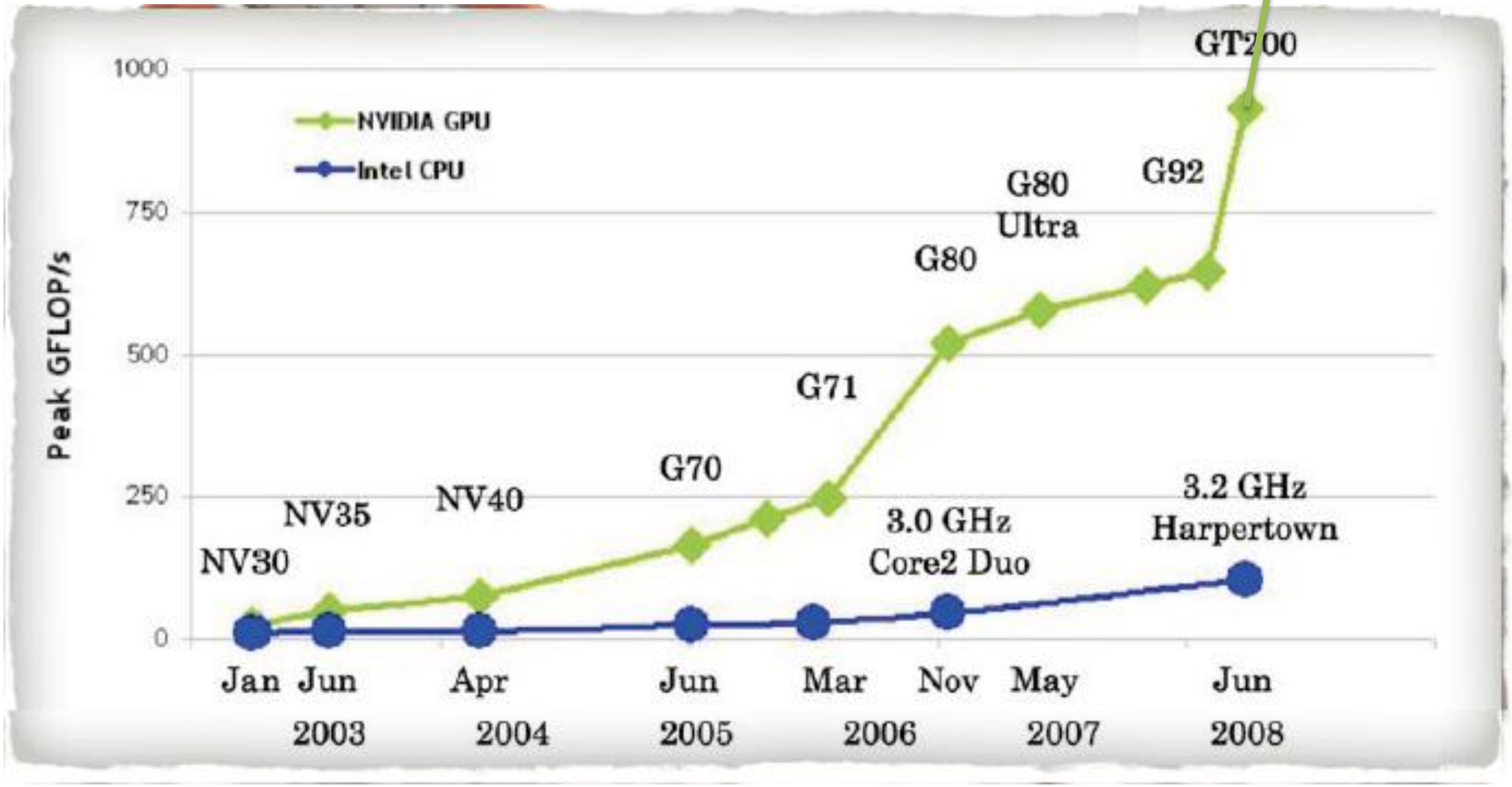
Computing power behind games

8 CORE CPU

480 CORE GPU

Video Card turn PC to HPC

GT300 “Fermi”



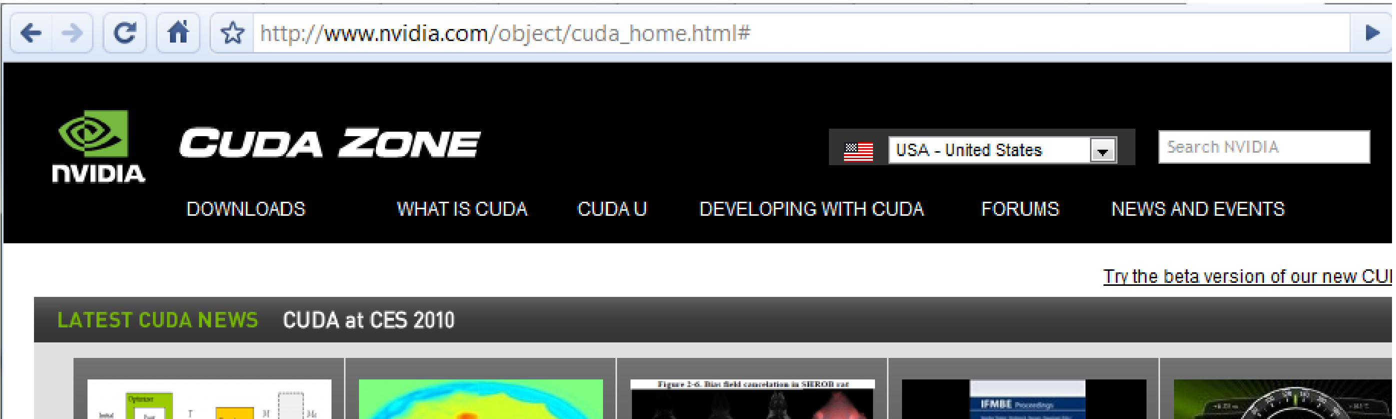
Super computer under your desk

$1.5 million

$1.5 K

Beyond Games

- GPU is ready for general purpose computing.
  - Accelerating operating system
  - Optimization tool box,
  - Linear algebra, FFT, LU decomposition…
  - Gene Folding
  - Physical Simulation
  - …



Sand box in VGC

A MULTI CAMERA CUDA ENABLED INTERACTIVE SYSTEM.

In VGC, you will…

- Learn Fresh New Technology
- Build FUN projects using super computers:
  - user computer interface
  - implement cutting edge vision algorithms
  - games
  - … NO Restriction!

In VGC, you will

- Learn visual processing tools.
- Get First hand projects experience
- Get in touch other geeks in BU.

Some rules in this club

- Interest come first.
- No bad ideas.
- Encourage brain storm. And select some to do.
- Learn by doing.
- … (TBA)

What do we do on meetings?

- Tutorial on techniques we need for projects.
- Discussion of new ideas.
- Show demos.
- Play games.
- Plan weekend projects.

Resources

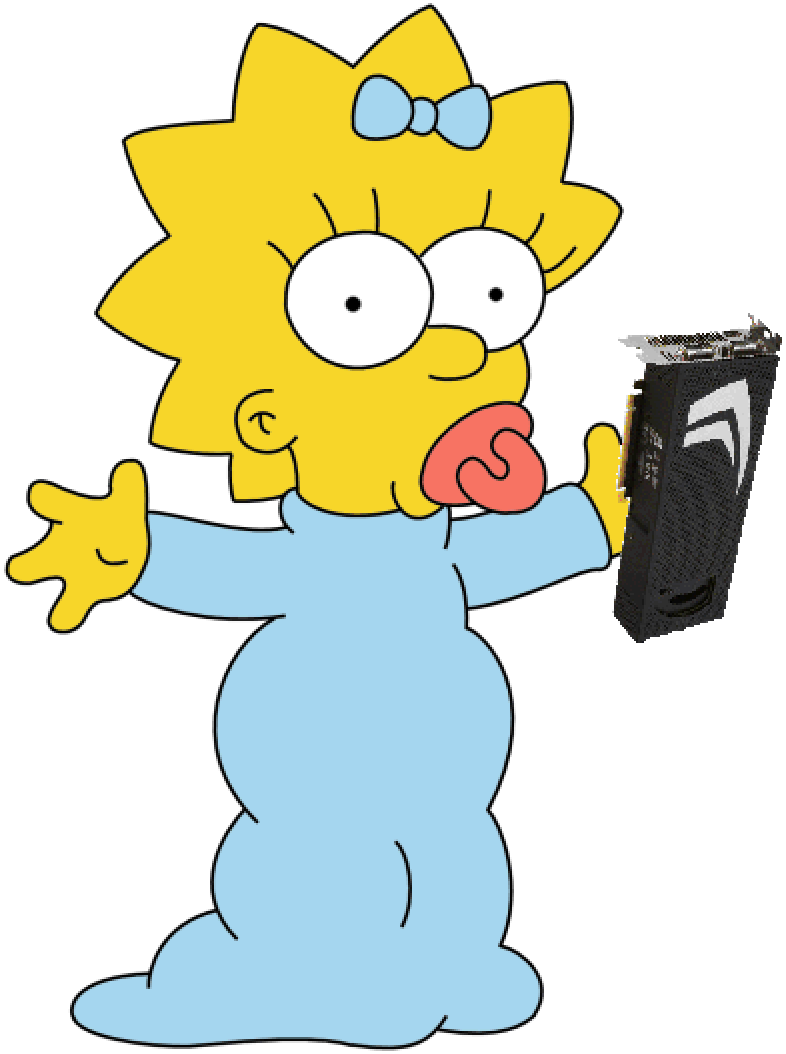
BU EC 500: High Performance Programming with Multi Core, GPU's (Herbordt, MW 2-4)

CUDA official website

MIT CUDA course: 6.963

VGC blog

VGC is young and need your help and participation



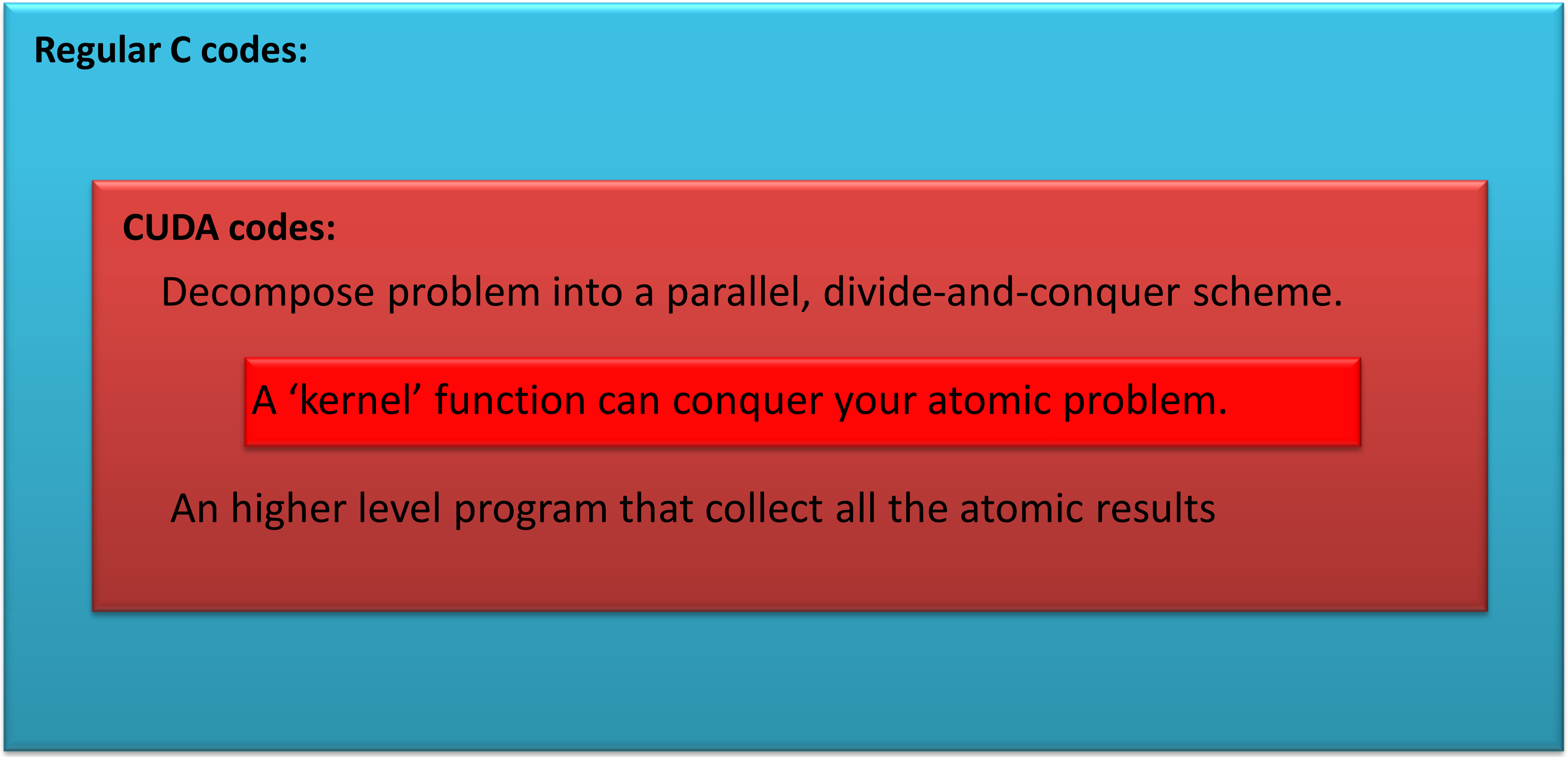
Weekend proj

Welcome to join us!

- This weekend:
- Home made multi touch interface

After the break, CUDA code walk though

CUDA Programming



Regular C codes:

CUDA codes:

Decompose problem into a parallel, divide-and-conquer scheme.

A ‘kernel’ function can conquer your atomic problem.

An higher level program that collect all the atomic results

Kernel Function

Kernel Function is the functions that actually run on each thread on GPU!

Write kernel functions in _kernels.cu file

Can be extremely simple:

PixelOperation_kernels.cu

```
__global__ void PixelOperation(char * * input, char * * output int w, int h);
```

Kernel Function

```
__global__ void PixelOperation(char * * input, char * * output int w, int h)

{
        //get the position on the image where t thread will work on
        short i = blockIdx.x*blockDim.x + threadId
        short j = blockIdx.y*blockDim.y + threadIdx.y

        //let each output pixel equal to the input
        output[i][j]= input[i][j];
}
```

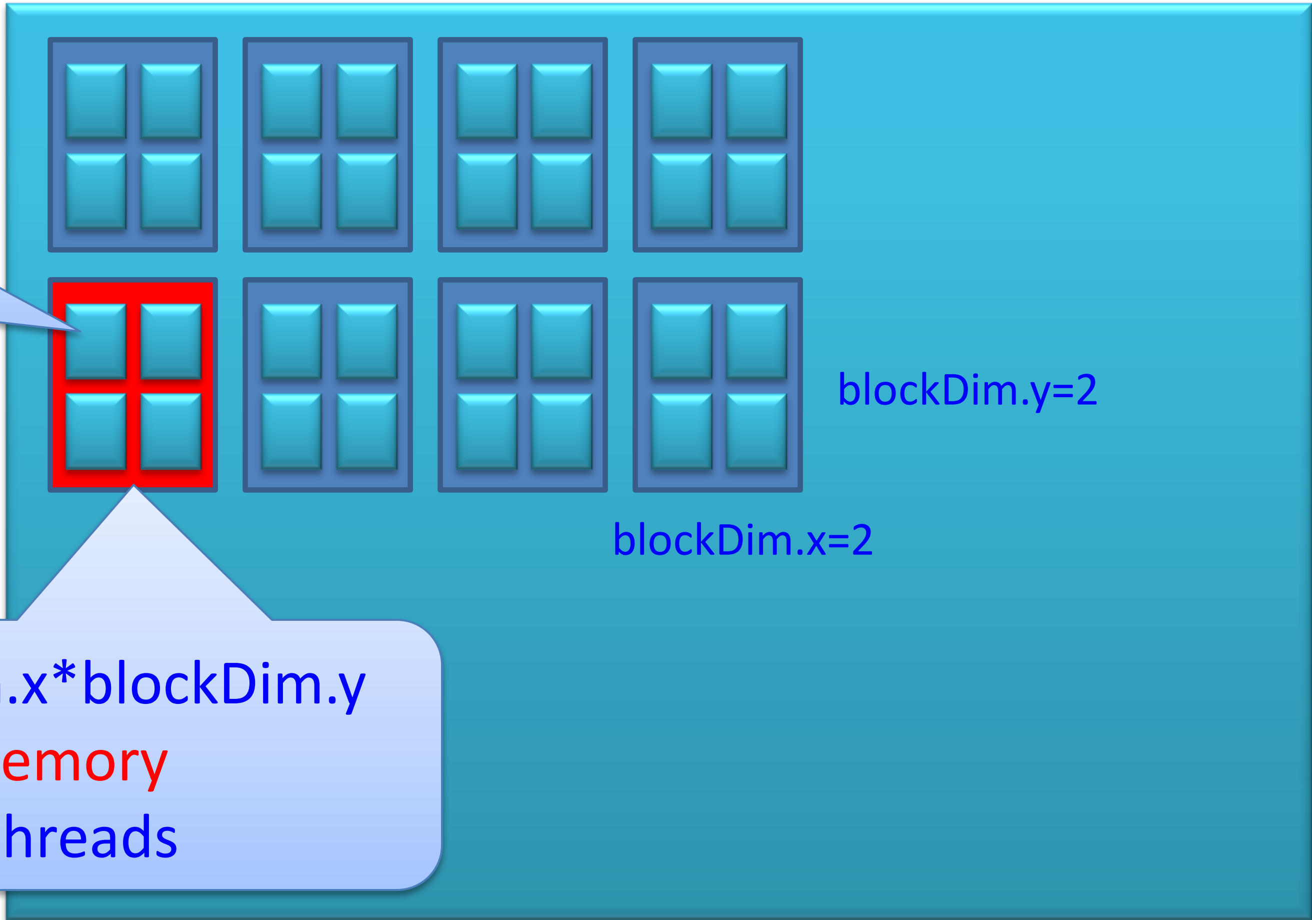
Pointer to the memory on the graphic card

This indicates this function is for GPU

The actual calculation on this thread.

Threads live in blocks

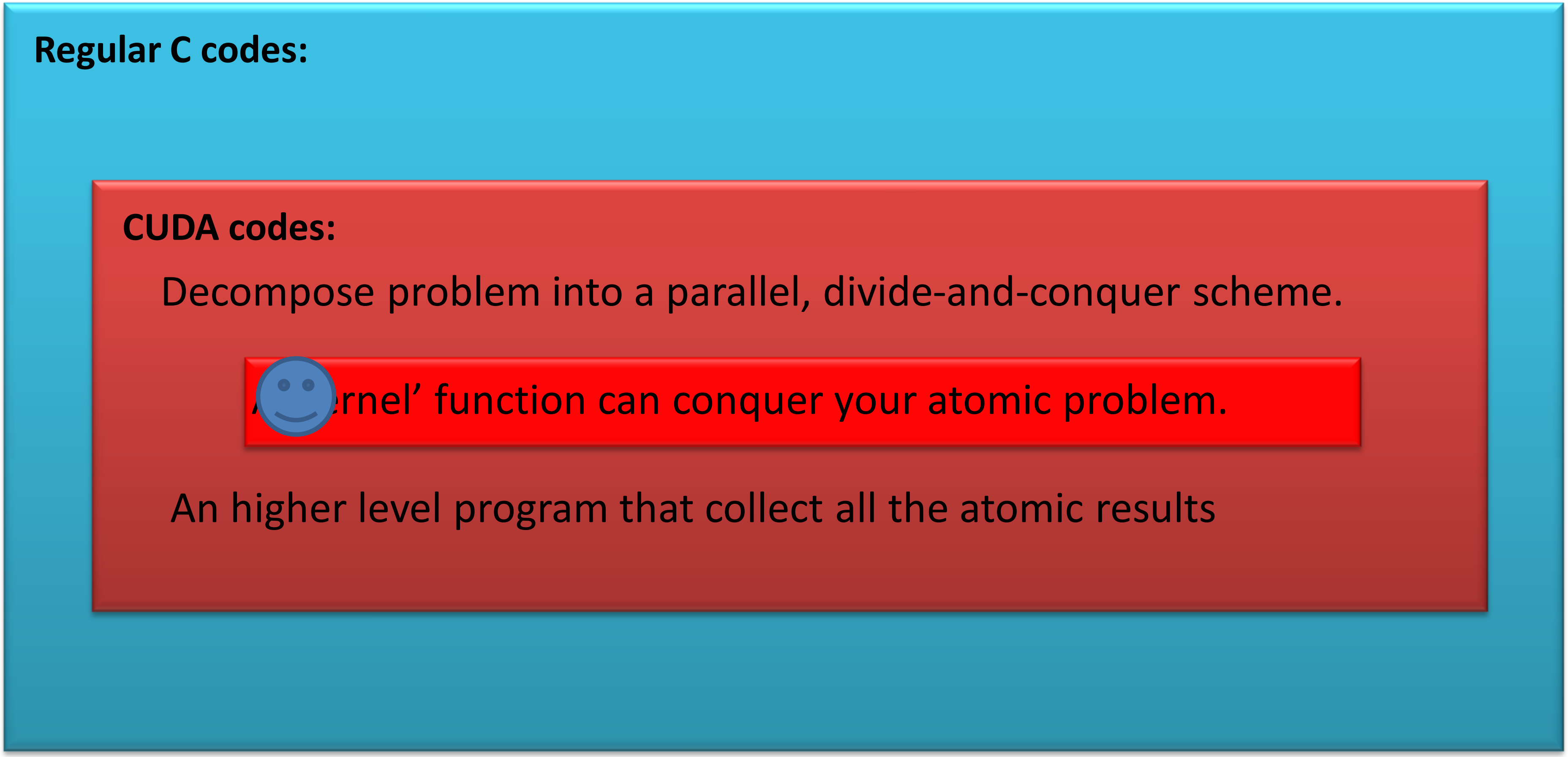
```
short i = blockIdx.x*blockDim.x + threadIdx.x; // horizontal position
short j = blockIdx.y*blockDim.y + threadIdx.y; // vertical position
```



Threads index:
threadIdx.x,
threadIdx.y

A block has blockDim.x*blockDim.y
threads, fast shear memory
In this case is 2x2=4 threads

CUDA Programming



Regular C codes:

CUDA codes:

Decompose problem into a parallel, divide-and-conquer scheme.

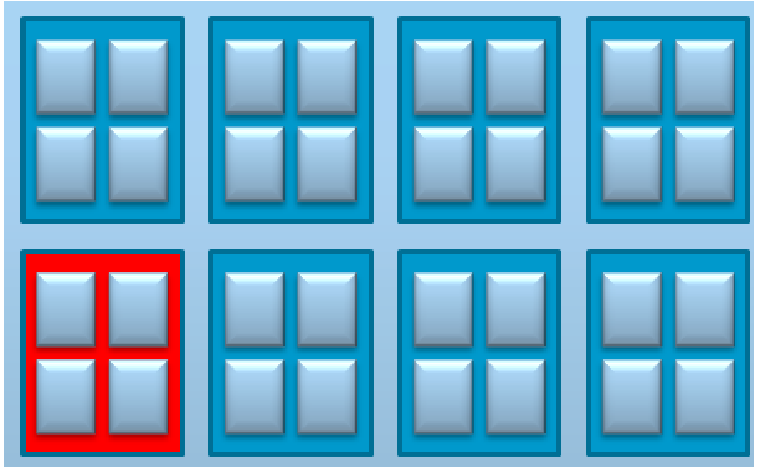
A ‘Kernel’ function can conquer your atomic problem.

An higher level program that collect all the atomic results

Call the kernel function

```
//do the calculation use kernel
dim3 dimBlock(2,2);
dim3 dimGrid(4, 2);

// Launch the device computation threads!
PixelOperation <<< dimGrid, dimBlock >>>  (input,outpur,width,height);
```



Provide the data!

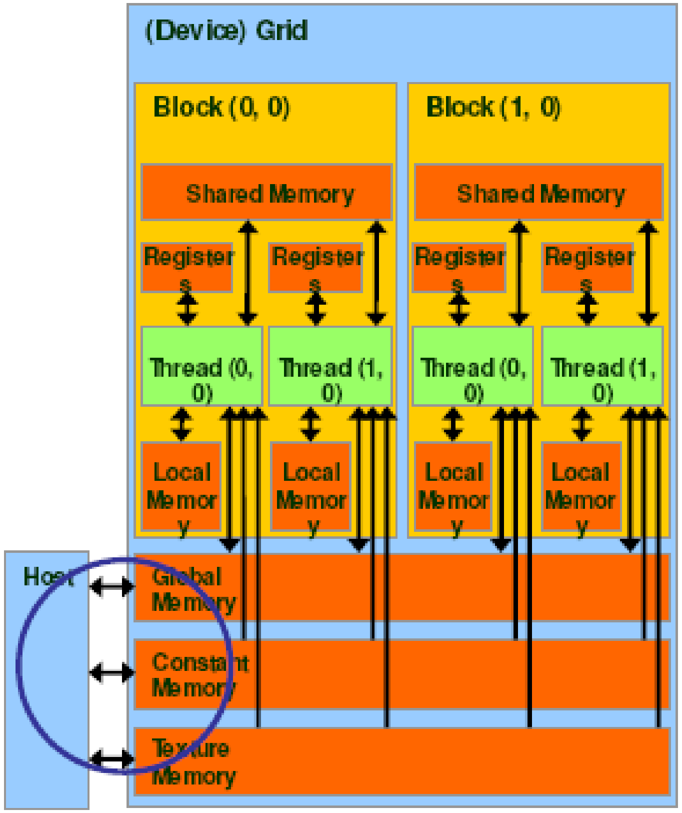
Graphic card need data to work on. So we give him.

In your program:
Create some space in the graphic card to receive your data

cudaMalloc (void ** pd, size);

Transfer your data using

cudaMemcpy(pd,p,size,
cudaMemcpyHostToDevice);



Launch the device computation threads!

PixelOperation <<< dimGrid, dimBlock >>> (pd,pd2,width,height);

Transfer your result back using

cudaMemcpy(result,pd2,size, cudaMemcpyDeviceToHost);

CUDA project prototype

Regular C code

```
int  main(void)
{
…

// do some stuff here!
output=imgBlur_Cuda(Char* input)


….
}
```

PixelOperation.cu file

```
Char* imgBlur_Cuda(Char* input)
{
        //Create some space in the graphic card
        to receive your data
        cudaMalloc (void ** pd, size);

        //Transfer your data using
        cudaMemcpy(pd,p,size,
        cudaMemcpyHostToDevice);

        //Launch the device computation threads!
        PixelOperation <<< dimGrid, dimBlock
        >>>  (pd,pd2,width,height);

        //Transfer your result back using
        cudaMemcpy(result,pd2,size,
        cudaMemcpyHostToDevice);

        Return result;
}
```

PixelOperation_kernel.cu file

```
__global__ void PixelOperation(char * *
input, char * * output int w, int h)

{
        //get the position on the image
        where this thread will work on
        short i = blockIdx.x*blockDim.x +
        threadIdx.x
        short j = blockIdx.y*blockDim.y +
        threadIdx.y;

        //let each output pixel equal to
        the input
        output[i][j]= input[i][j];
}
```