Heat Equation

Francesco Pasa
Enrico Panontin

Technische Universität München
Physics Department
Parallelisation of Physics Calculations on GPUs with CUDA

16 Juni 2016

Demonstration

We consider heat propagation in a medium (e.g. a solid) far away from any state transition and we define:

ϵ : internal energy density

c : specific heat

ρ : mass density

T : temperature

By virtue of the first thermodinamic principle ($\Delta U = \Delta Q - L = \Delta Q$):

$$d\left[\int_{\Omega} c\rho T \, \mathrm{d}V\right] = \left[-\int_{\partial\Omega} \vec{J} \cdot \vec{n} \, \mathrm{d}S\right] \mathrm{d}t \tag{1}$$

Demonstration

$$d\left[\int_{\Omega} c\rho T \,\mathrm{d}V\right] = \left[-\int_{\partial\Omega} \vec{J}\cdot\vec{n}\,\mathrm{d}S\right]\mathrm{d}t$$

Newton-Fourier Law, heat spreads and temperature becomes uniform throughout the medium:

$$\vec{J} = -k\vec{\nabla}T \tag{2}$$

By substituting equation (2) in (1) and applying the *divergence theorem* one gets the **heat equation**:

$$\frac{\partial T}{\partial t} - \frac{k}{c\rho}\Delta T = 0 \tag{3}$$

Heat Equation

$$\frac{\partial T}{\partial t} - \frac{k}{c\rho}\Delta T = f(\vec{x}, t)$$

- First order in time
- Second order in space
- If we consider a heating system, we must add $f(\vec{x}, t)$

Numerical Solution

$$\frac{\partial T}{\partial t} - \frac{k}{c\rho}\Delta T = f(\vec{x}, t)$$

Discretize space and time: foreward difference for time, central difference for space:

$$T_{i,j}^{n+1} = T_{i,j}^{n} + \eta\left[T_{i+1,j}^{n} + T_{i-1,j}^{n} + T_{i,j+1}^{n} + T_{i,j-1}^{n} - 4T_{i,j}^{n}\right] \quad (4)$$

where n is the time index, i, j are x-index and y-index, $\eta = \frac{k\Delta t}{c\rho(\Delta x)^2}$.

General Structure

sim2d.cu

integrator.h
integrator.cu

gl_helper.cu
gl_helper.h

sim2d.cu

Set: number of block, number of threads, other parameters.

```
void readTiff(char *filename, float **raster, unsigned *w,
              unsigned *h, float scale)
void step()

glutMainLoop()
```

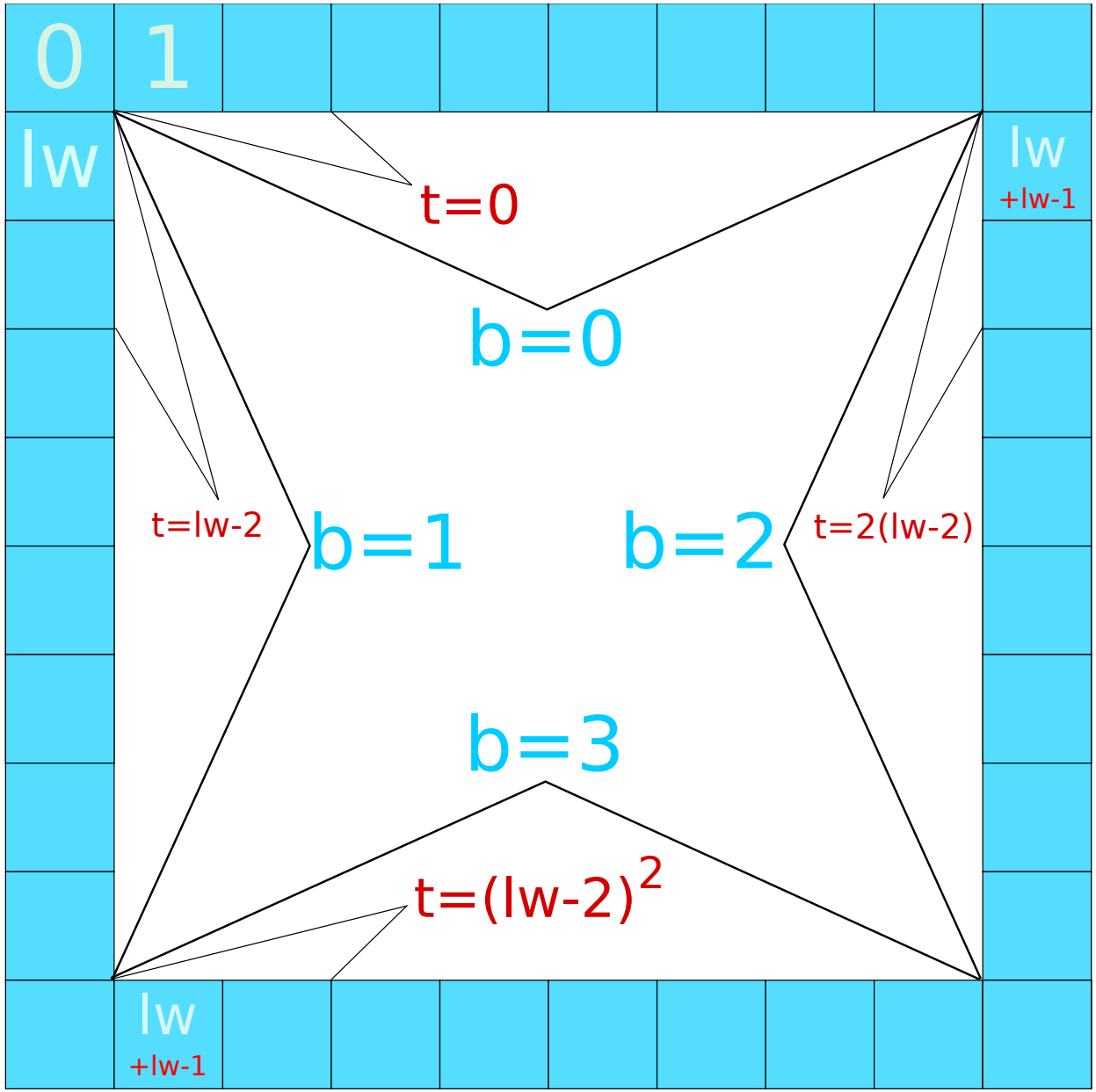
integrator.cu

```
__global__
void stepSimulation2D(float *T, float *K, float *dT,
                      unsigned n_loop, uchar4 *tex,
                      char copy_tex)
__device__
void loadSharedMemory2D(const UsefulConstants consts,
                      float *T)
__device__
void integrate2D(const UsefulConstants consts, float *T,
                      float *K, float *dT)
```

loadSharedMemory2D

```
for (i = 0; i < n_loop; ++i) {
        local_T[lid_1d + i] = T[gid_1d + i];
        //d_operation[gid_1d+i] = 255;
}
__syncthreads();
```

loadSharedMemory2D



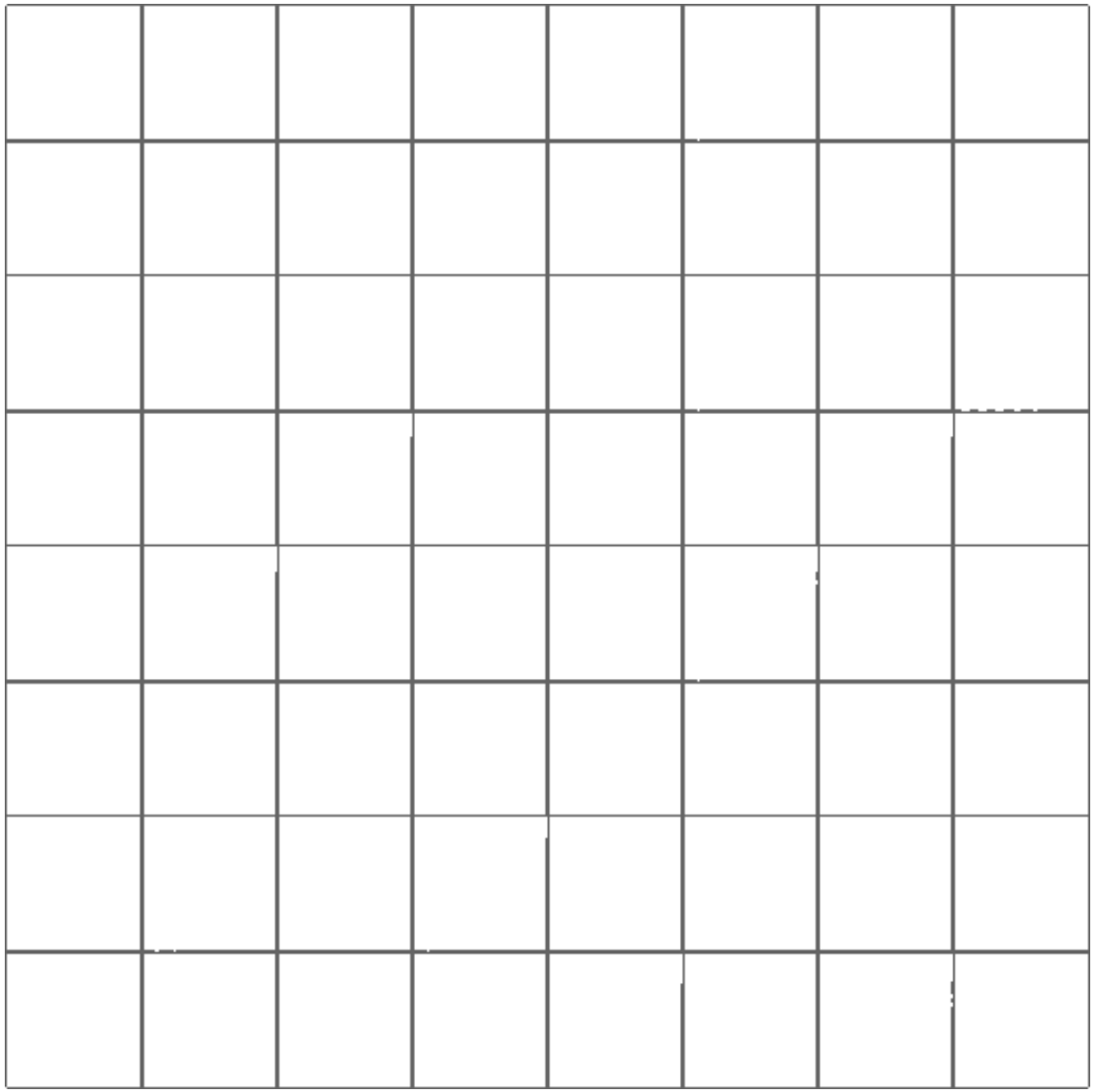
integrate2D

```
for (i = 0; i < consts.n_loop; ++i) {
        T[gid_1d+i] += K[gid_1d_nb+i] *
                (local_T[lid_1d+1+i] + local_T[lid_1d-1+i]
                + local_T[lid_1d+lw+i] + local_T[lid_1d-lw+i]
                - 4*local_T[lid_1d+i])

                + dT[gid_1d_nb+i];
}
```

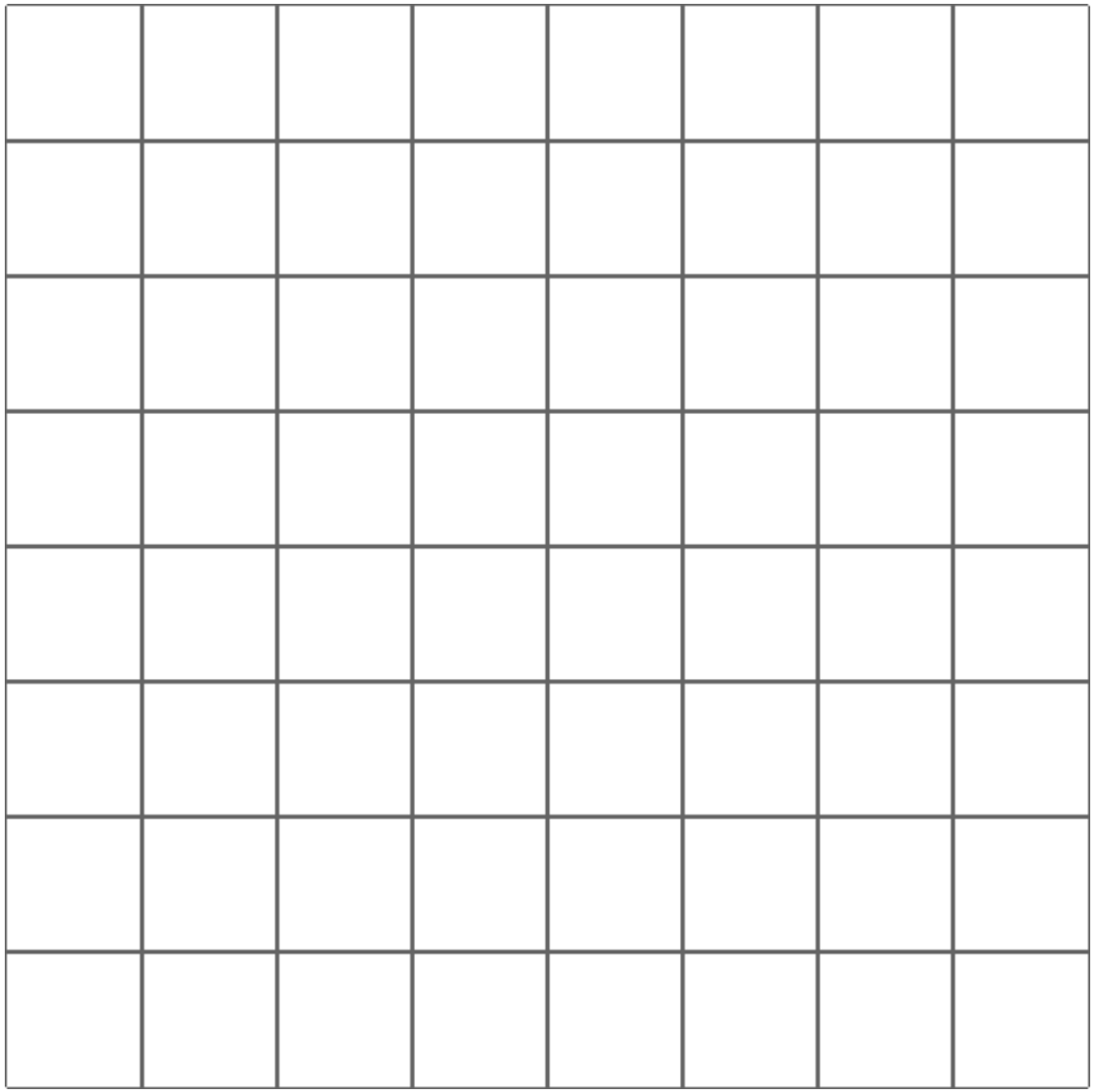
Shared Memory

Random errors



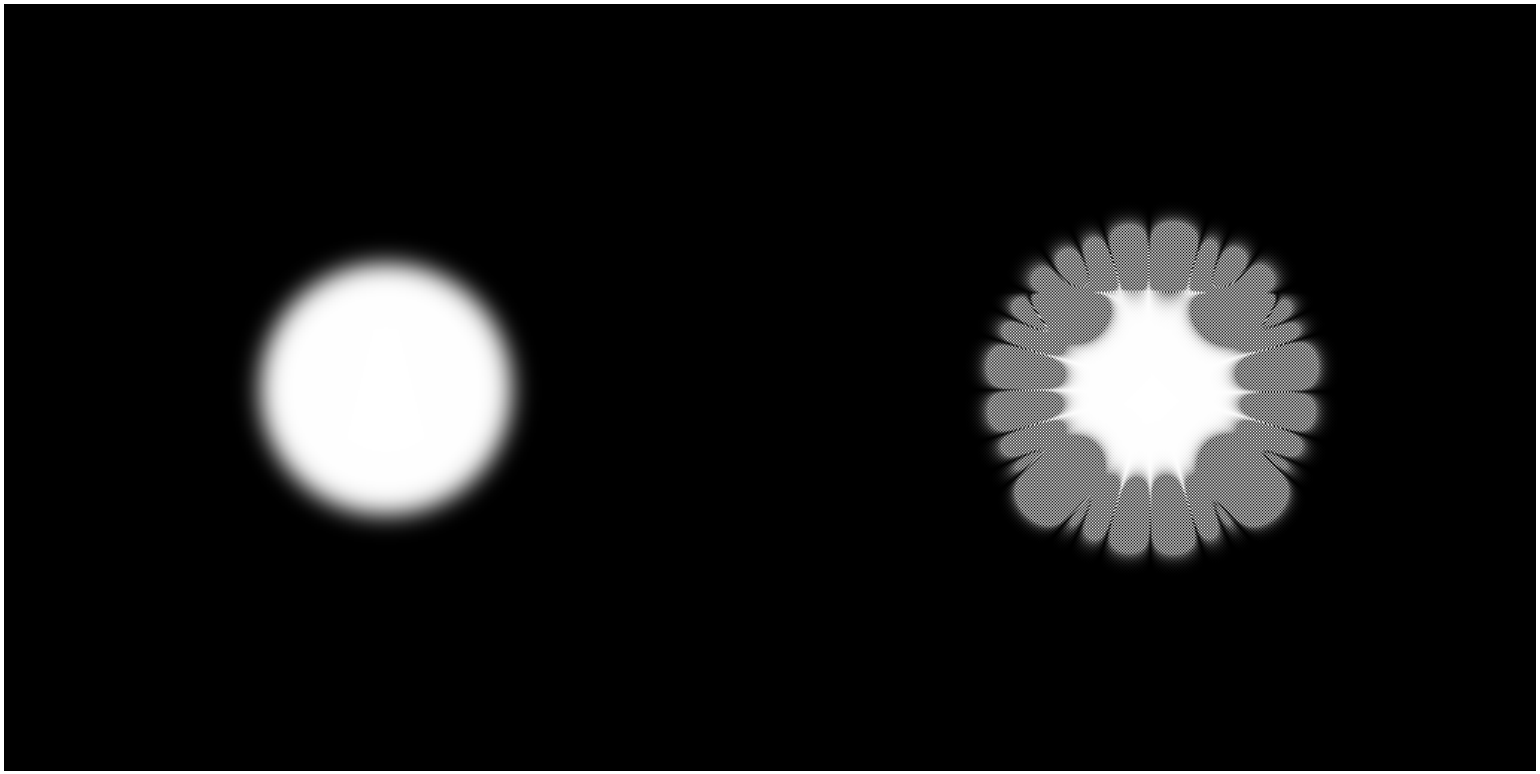
Shared Memory

Synchronized threads



Stability

$$\Delta t = 0.001 \longrightarrow \Delta t = 0.002$$



Performances

GPU

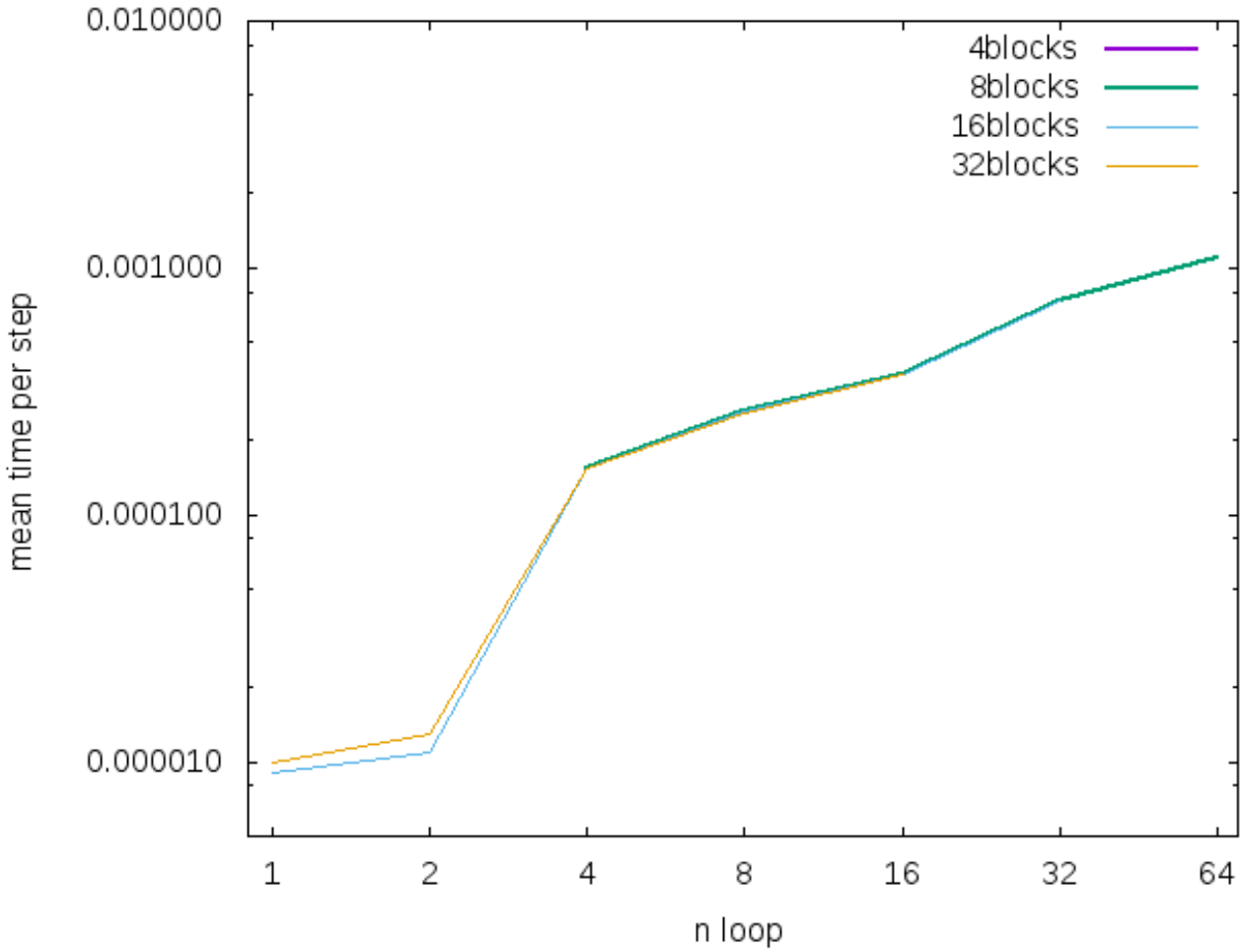
```
Device 0: "GeForce GTX 780"

CUDA Driver Version / Runtime Version          8.0 / 6.5
CUDA Capability Major/Minor version number:    3.5
Total amount of global memory:                 3071 MBytes
(12) Multiprocessors, (192) CUDA Cores/MP:     2304 CUDA Cores
GPU Clock rate:                                941 MHz
Memory Clock rate:                             3004 Mhz
Memory Bus Width:                              384-bit
L2 Cache Size:                                 1572864 bytes

Total amount of constant memory:               65536 bytes
Total amount of shared memory per block:       49152 bytes
Total number of registers available per block: 65536
Warp size:                                     32
Maximum number of threads per multiprocessor:  2048
Maximum number of threads per block:           1024
```

Performances

Execution time



Performances

Execution time GPU vs CPU

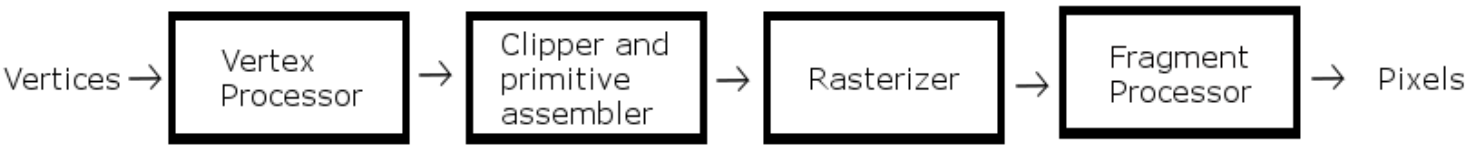
Performances

Different GPUs

OpenGL

OpenGL is a graphics API (similar in scope to Direct3D) that allows to use the graphics card to draw 2D and 3D vector graphics.

- 1992 - OpenGL is first released
- 2004 - OpenGL 2.0 is released. Adds support for programmable pipeline (shaders).
- 2010 - OpenGL 4.0 is released. Adds support for geometry shader (tesselation).



Old method for drawing

```
glColor3f(1.0f, 0.0f, 0.0f);
glBegin(GL_QUADS);
    glVertex2f(-0.25f, 0.25f);
    glVertex2f(-0.5f, -0.25f);
    glVertex2f(0.5f, -0.25f);
    glVertex2f(0.25f, 0.25f);
glEnd();
```

Moreover the API had to support many features, for example textures coordinates, lighting, shadows, coordinate transformation and perspective matrices.

This results in a complex API and lack of flexibility.

Modern approach

OpenGL state machine

Since the rendering of the 3D scene is a very complex task, OpenGL uses the concept of state machine to simplify the API interface (avoid function with too many arguments).

```
// Tell OpenGL which array contains the data
glBindBuffer(GL_ARRAY_BUFFER, vbo);
// Specify how the data for position can be accessed
glVertexAttribPointer(0, size, GL_FLOAT, GL_FALSE, 0, 0);
// Enable the attribute
glEnableVertexAttribArray(0); // location = 0

// Draw
glDrawArrays(type, 0, vertex_num);
```

OpenGL VBOs

VBO (Vertex Buffer Object) - is an array of data in the GPU memory for storing vertices.

OpenGL Textures

Texture - is an image that is mapped to vertices.