

Lattice simulation on graphics cards



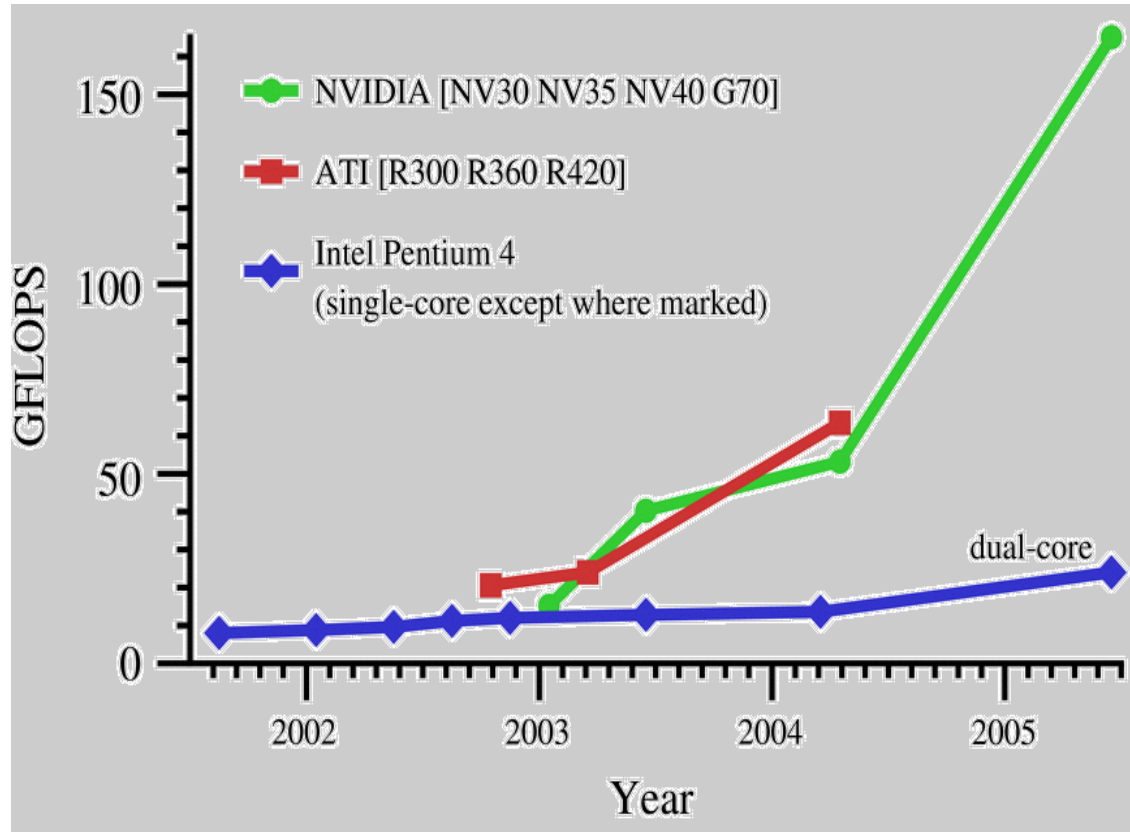
Dániel Nógrádi

in collaboration with

Győző Egri, Zoltán Fodor, Christian Hoelbling
Sándor Katz, Kálmán Szabó

University of Wuppertal – Eötvös University Budapest

Why use Graphical Processing Units?

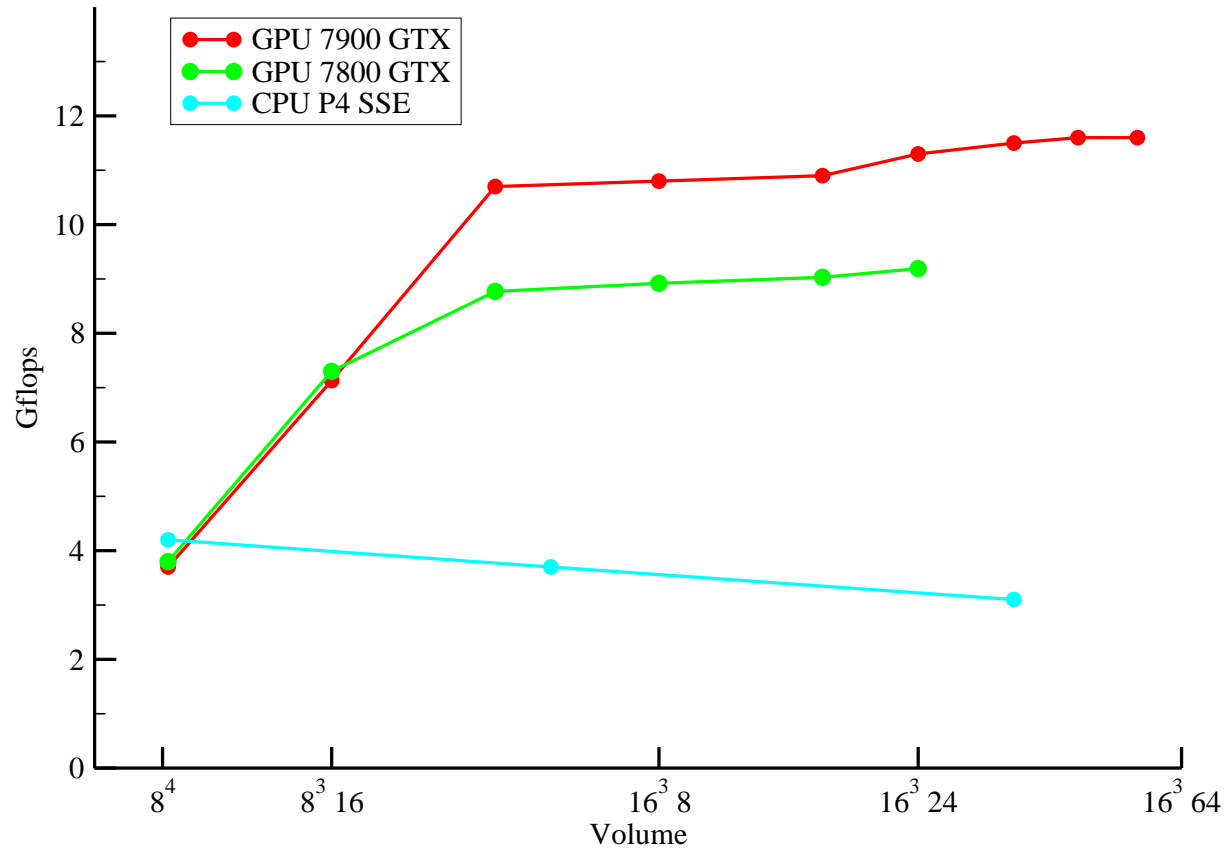


Ian Buck, Stanford

Gamerz market is big \$\$\$: top model costs ~ \$500

How much power can be utilized for the lattice?

Performance for Wilson multiplication



Conjugate gradient: 90% of Wilson \sim 10 Gflops

How is this possible?

- CPU: lot of transistors do non-computational tasks
- GPU: transistors (almost) only calculate

GPU architecture

- Native data: 2D arrays, textures (for games its content ends up on the screen).
- Each pixel (or fragment): 4 floating point numbers: RGBA colour channels.
- Each pixel computation: incoming textures → outgoing textures: same operation on each pixel → massively parallel.
- Like a stream: pipeline should be full → performance is better for large textures (lattices).

Software

- Basic framework to interact with GPU from the CPU is OpenGL.
- OpenGL is a graphics library + extensions which are tailored for the given GPU (typically invented by the GPU vendors).
- In C source code one calls OpenGL functions that manage textures on the GPU (create texture, upload/download data, etc).
- Actual computation on the textures are done by pixel shaders.
- Pixel shaders can be written in an assembly-like language or a higher level language called Cg (resembling C) which gets compiled to assembly.

Real life example: $x_i = y_i + z_i \quad i = 1 \dots 4nm$

Shader in Cg

```
struct FragmentOut { float4 color0:COLOR0; };
```

```
FragmentOut example( in float2 myTexCoord:WPOS,  
    uniform samplerRECT y, uniform samplerRECT z )  
{  
    FragmentOut c;  
    c.color0 = texRECT( y, myTexCoord ) + texRECT( z, myTexCoord );  
    return c;  
}
```

Create texture in OpenGL (like malloc)

```
GLuint X;  
glGenTextures( 1, &X);  
glBindTexture( ..., X);  
glTexParameteri(...);  
glTexImage2D( ..., n, m, ..., 0 );           // do this for Y, Z also
```

Transfer data CPU → GPU in OpenGL

```
glBindTexture( ..., Y);  
glFramebufferTexture2D( ..., Y, ... );  
glTexSubImage2D( ..., n, m, ..., y );       // do this for z also
```

Do the actual computation, run the Cg shader

```
cgGLSetTextureParameter( ..., Y);
cgGLEnableTextureParameter( ..., "y" );           // also for z
glFramebufferTexture2DEXT( ..., X, 0 );
glDrawBuffer( ... );
cgGLBindProgram( ... );
glBegin( ... );
{
    glVertex2f( -n, -m);
    glVertex2f(  n, -m);
    glVertex2f(  n,  m);
    glVertex2f( -n,  m);
}
glEnd( );
```

Transfer data GPU → CPU in OpenGL

```
glFramebufferTexture2D( ..., X, ... );  
glReadBuffer( ... );  
glReadPixels( ..., n, m, ..., x);
```

Same in C:

```
for( i = 0; i < 4 * n * m; i++ ) x[i] = y[i] + z[i];
```

Even more real life example: Wilson code

- We currently have production code for dynamical staggered and overlap simulations running on Nvidia 7800 GTX and 7900 GTX.
- A lattice of size $N_x N_y N_z N_t$ is represented by $N_x N_y \times N_z N_t$ textures and so one site corresponds to one pixel.
- A half-wilson vector ψ_a^i is a complex vector with $i = 1, 2$ half-dirac and $a = 1, 2, 3$ colour index.
- Each colour component has 4 real components which exactly fits into a pixel and so a half-wilson vector is put into 3 textures.

Wilson code continued: Gauge links

- 18 real components for fixed direction
- Need 5 textures and there is room for $2 = 5 \times 4 - 18$ more real numbers.
- Store 2D texture location of its neighbour.
- Over all the gauge links are put into 20 textures.

Difficulties

- Complicated (better: unusual) programming model
- Single precision, not a problem since CPU is still there with double precision.
- Relatively small memory, currently 512 MB at best

Future prospects

- Communication between the cards (in one PC or different PC's)
- Software side: unknown and unpredictable progress made by vendors resulting in new OpenGL extensions and improved drivers.
- Hardware side: faster, larger, more!