GPU-based high-performance computing for radiotherapy applications

Julien Bert, PhD

CHRU de Brest
LaTIM – INSERM UMR1101
Radiotherapy

Irradiation of the tumor:
- Maximum dose to the tumor
- Healthy surrounding organs spared

External Beam Radiotherapy

Aims:
- Predict the dose
- Personalized planning
- Ensure quality and safety

Accurate and realistic numerical simulation in radiation physics

Intra-Operative Radiotherapy
Monte Carlo method

Estimated the value of $\pi$

$\pi \approx 3.16967$

Uniformly scatter some objects of uniform size (rice or sand) over the square

Monte Carlo Simulation

Random sample must follow the laws of physics

Simulate particle interactions with matter

Monte Carlo Simulation
Monte Carlo simulation

Very computationally demanding
limits their applicability in both research and clinical environment applications

Computer cluster
financial burden and availability issues

Cost
Space

• Patient
• Workflow
• Workspace
• Intervention cost

h-GATE (2010)
Computational power

4 graphics cards \( \times 2 \) GPUs\(^1\) vs.

249 processors\(^2\) with \( \times 6 \) cores (1494 cores)

High computational power at a reduce cost (1/20)

---

\(^1\) NVIDIA GTX Titan Z = 8.1 TFLOPS
\(^2\) Intel Core i7 980 x6 3.33 GHz = 130 GFLOPS
Programming language

Writing a “non-graphics” program with a graphics API

2003 Cg NVIDIA (Mark et al. ACM Trans. Graph.)

2004 Brook ATI (Buck et al. SIGGRAPH)

2004 OpenGL shading language (Kessenich et al. http://www.opengl.org)

2007 CUDA NVIDIA (Compute Unified Device Architecture)
Monte Carlo simulation on GPU

Paradigm
- one thread per history
- particles are independent
- easy to parallelize

Thousands of particles are simulated in parallel
Divergence

If statement

Begin

If

A  photon

B  electron

End

Threads

Parallel programming  Divergence

37e Forum ORAP – march 2016
Geant4 is a toolkit for the simulation of the passage of particles through matter. Its areas of application include high energy, nuclear and accelerator physics, as well as studies in medical and space science. The two main reference papers for Geant4 are published in *Nuclear Instruments and Methods in Physics Research* A 506 (2003) 250-303, and *IEEE Transactions on Nuclear Science* 53 No. 1 (2006) 270-278.

Applications

- A *sampling of applications, technology transfer and other uses of Geant4*

User Support

- *Getting started, guides and information for users and developers*

Publications

- *Validation of Geant4, results from experiments and publications*

Collaboration

- *Who we are: collaborating institutions, members, organization and legal information*

- Geant4 code on GPU (C++ ➔ C ➔ CUDA)
## Types of memory

<table>
<thead>
<tr>
<th>Storage type</th>
<th>Register</th>
<th>Shared memory</th>
<th>Texture memory</th>
<th>Constant memory</th>
<th>Global memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bandwidth</strong></td>
<td>~8 TB/s</td>
<td>~1.5 TB/s</td>
<td>~200 MB/s</td>
<td>~200 MB/s</td>
<td>~200 MB/s</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>1 cycle</td>
<td>1 to 32 cycles</td>
<td>~400 to 600</td>
<td>~400 to 600</td>
<td>~400 to 600</td>
</tr>
<tr>
<td><strong>Lifetime</strong></td>
<td>Function</td>
<td>kernel</td>
<td>application</td>
<td>application</td>
<td>application</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>Thread</td>
<td>Block</td>
<td>All threads</td>
<td>All threads</td>
<td>All threads</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Read/ Write</strong></td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
</tr>
<tr>
<td><strong>Cached</strong></td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

- **limited variables**
- **geometry and physical tables**
- **physical constants**
- **particles**
Pseudo Random Generator Number (PRNG)

Return a random number uniformly distributed between 0 and 1

Need to store generator state

Find a suitable algorithm:
- small state size
- large period
Simulation loop

14 billions of particles

~ 2-4 GB

Source kernel

Particles buffer

Simulation Kernel (all steps)

Main simulation loop

Total number of particles requested?

No

Yes

exit
Array of Structure (AoS)

```c
struct Point {
    float x;
    float y;
    float z;
};
```

- list of points
- regular access is **not optimized**

Structure of Array (SoA)

```c
struct Point {
    float *x;
    float *y;
    float *z;
};
```

- Point contains a list of x, y, z values
- coalesced access is **optimized**

Particle data structure

```c
struct StackGamma{
    float* E;
    float* dx;
    float* dy;
    float* dz;
    float* px;
    float* py;
    float* pz;
    float* t;
    unsigned int* seed;
    unsigned char* interaction;
    unsigned char* live;
    unsigned char* endsimu;
    unsigned char* ct_cpt;
    unsigned char* ct_pe;
    unsigned char* ct_ray;
    unsigned int size;
    unsigned long* table_x_brent;
}; //
```
Monte Carlo simulation on GPU (2012):

- **Pseudo random number generator**
- Electromagnetic effects for *photon*
  - Compton scattering
  - Rayleigh scattering
  - Photoelectric effect
- Voxelized *geometry navigation*
- *Materials properties*
- Single precision (float number)

2013 - Full agreement between **GPU code** and Geant4 (**CPU**)

Geant4-based Monte Carlo simulations on GPU for medical applications

Julien Bert\textsuperscript{1,5}, Héctor Perez-Ponce\textsuperscript{2,5}, Ziad El Bitar\textsuperscript{3}, Sébastien Jan\textsuperscript{4}, Yannick Boursier\textsuperscript{2}, Damien Vintache\textsuperscript{3}, Alain Bonissent\textsuperscript{2}, Christian Morel\textsuperscript{2}, David Brasse\textsuperscript{3} and Dimitris Visvikis\textsuperscript{1}

\textsuperscript{1} LaTIM, UMR 1101 INSERM, CHRU Brest, Brest, France
\textsuperscript{2} CPPM, Aix-Marseille Université, CNRS/IN2P3, Marseille, France
\textsuperscript{3} IPHC, UMR 7178—CNRS/IN2P3, Strasbourg, France
\textsuperscript{4} DSV/I2BM/SHFJ, Commissariat à l’Energie Atomique, Orsay, France
Bugs

Tracking bugs:
- `printf` (device code, cuda 3.1 2010)
- Nsight debugger

Nsight Debugger
- Seamless and simultaneous debugging of both CPU and GPU code
- View program variables across several CUDA threads
- Examine execution state and mapping of the kernels and GPUs
- View, Navigate and filter to selectively track execution across threads
- Set breakpoints and single-step execution at both source-code and assembly levels
- Includes cuda-memcheck to help detect memory errors

```c
__device__ __host__ void MyFunction()
```
Science applications

Floating point IEEE 754 (single precision)

Double precision available on GPU:
- slower computation

Navigation (single precision)

Dose deposition (double precision)

very small (dE) + very high (Tot E) = no change (Tot E)
Clinical context

2015 - Radiotherapy (brachytherapy)

Radiotherapy (external beam)

GGEMS-Brachy: GPU GEant4-based Monte Carlo simulation for brachytherapy applications

Yannick Lemaréchal¹, Julien Bert¹, Claire Falconnet¹, Philippe Desprès¹,²,³, Antoine Valeri¹,², Ulrike Schick¹,², Olivier Pradier¹,², Marie-Paule Garcia¹, Nicolas Boussion¹,² and Dimitris Visvikis¹
**GPU GEant4-based Monte Carlo Simulations**

http://www.ggems.fr

GGEMS is a library not a software.
### Heterogeneous architecture

#### Technical Specifications

<table>
<thead>
<tr>
<th>Feature</th>
<th>Compute Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of 32-bit registers per multiprocessor</td>
<td>8 K</td>
</tr>
<tr>
<td>Maximum number of 32-bit registers per thread</td>
<td>128</td>
</tr>
<tr>
<td>Maximum amount of shared memory per multiprocessor</td>
<td>16 KB</td>
</tr>
</tbody>
</table>

#### Feature Support

<table>
<thead>
<tr>
<th>Feature</th>
<th>Compute Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warp vote functions (Warp Vote Functions)</td>
<td></td>
</tr>
<tr>
<td>Double-precision floating-point numbers</td>
<td>No</td>
</tr>
<tr>
<td>Atomic functions operating on 64-bit integer values in shared memory (Atomic Functions)</td>
<td></td>
</tr>
<tr>
<td>Atomic addition operating on 32-bit floating point values in global and shared memory (atomicAdd())</td>
<td>No</td>
</tr>
</tbody>
</table>

GTX580 cc 2.0 512 cores (Fermi)
GTX680 cc 3.0 1536 cores (Kepler)
GTX Titan cc 3.5 2688 cores (Kepler)
Heterogeneous architecture

CUDAHOME := /usr/local/cuda/lib64
CUDALIBS := -L$(CUDAHOME)/lib -lcudart
CUDAFLAGS := -gencode=arch=compute_30,code=sm_30 \ 
    -gencode=arch=compute_32,code=sm_32 \ 
    -gencode=arch=compute_35,code=sm_35 \ 
    -gencode=arch=compute_37,code=sm_37 \ 
    -gencode=arch=compute_50,code=sm_50 \ 
    -gencode=arch=compute_52,code=sm_52 \ 
    --compiler-options -w --std c++11
Heterogeneous architecture

Hardware evolution:

Hardware upgrade have a cost!

CUDA, new features:
- updating the code
- using new graphics cards

Plug, compile and play

Instantly speed up your application
Main CUDA code is similar to the original one (C++)

```c
// Compton Cross Section Per Atom (Standard - Klein-Nishina)
__device__ float Compton_CSPM(float E, unsigned short int Z) {
    float CrossSection = 0.0;
    if (Z < 1 || E < 1e-4f) {
        return CrossSection;
    }
    float p1Z = Z*(2.7956e-23f + 1.9756e-27f*Z + 3.9178e-29f*Z*Z);
    float p2Z = Z*(-1.8308e-23f + 1.0206e-27f*Z + 6.241e-29f*Z*Z);
    float p3Z = Z*(6.7531e-22f + 7.2913e-24f*Z + 6.048e-27f*Z*Z);
    float p4Z = Z*(-1.9756e-21f + 2.0708e-24f*Z + 3.0275e-26f*Z*Z);
    float T0 = (Z < 1.5f) ? 40.0e-3f : 15.0e-3f;
    float d1, d2, d3, d4, d5;

    d1 = __fdxdividef(E, T0, 0.510998910f); // X
    CrossSection = __fdxdividef(p1Z*__logf(1.0f-2.0f*d1), d1) + __fdxdividef(p2Z + p3Z*d1 + p4Z*d1*d1, 1.0f +
                20.0f*d1 + 230.0f*d1*d1 + 440.0f*d1*d1*d1); // Y

    if (E < T0) {
        d1 = __fdxdividef(T0-1.0e-3f, 0.510998910f); // X
        d2 = __fdxdividef(p1Z*__logf(1.0f+2.0f*d1), d1) + __fdxdividef(p2Z + p3Z*d1 + p4Z*d1*d1, 1.0f + 20.0f*d1 +
                230.0f*d1*d1 + 440.0f*d1*d1*d1); // Y
        d3 = __fdxdividef(-T0* (d2 - CrossSection), CrossSection*1.0e-3f); // c1
        d4 = (Z > 1.5f) ? 3.975f-0.0556f*__logf(Z) : 0.15f; // c2
        d5 = __logf(__fdxdividef(E, T0)); // y
        CrossSection = __expf(-d3 + (d4 + d5)*d5));
    }

    return CrossSection;
}
```
Conclusion

GPU-Accelerated Libraries

- cuFFT – Fast Fourier Transforms Library
- cuBLAS – Complete BLAS library
- cuSPARSE – Sparse Matrix library
- cuRAND – Random Number Generator
- NPP – Thousands of Performance Primitives for Image & Video Processing
- Thrust – Templated Parallel Algorithms & Data Structures
- CUDA Math Library of high performance math routines

- PhysX – scalable multi-platform game physics solution
- VisualFX – solutions for complex, cinematic visual effects
- OptiX – Fast ray tracing engine

NVIDIA®
Conclusion

Specifications:

<table>
<thead>
<tr>
<th>Number of 32-bit registers per multiprocessor</th>
<th>8 K</th>
<th>16 K</th>
<th>32 K</th>
<th>64 K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of 32-bit registers per thread</td>
<td>128</td>
<td>63</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Maximum amount of shared memory per multiprocessor</td>
<td>16 KB</td>
<td>48 KB</td>
<td>64 KB</td>
<td></td>
</tr>
</tbody>
</table>

Developer friendly:
- debugging
- thread exception error
Conclusion

- Computer assisted medical intervention
- Image-guide radiotherapy
- Treatment planning
- ...
Questions?

Julien Bert, PhD

CHRU de Brest
LaTIM – INSERM UMR1101