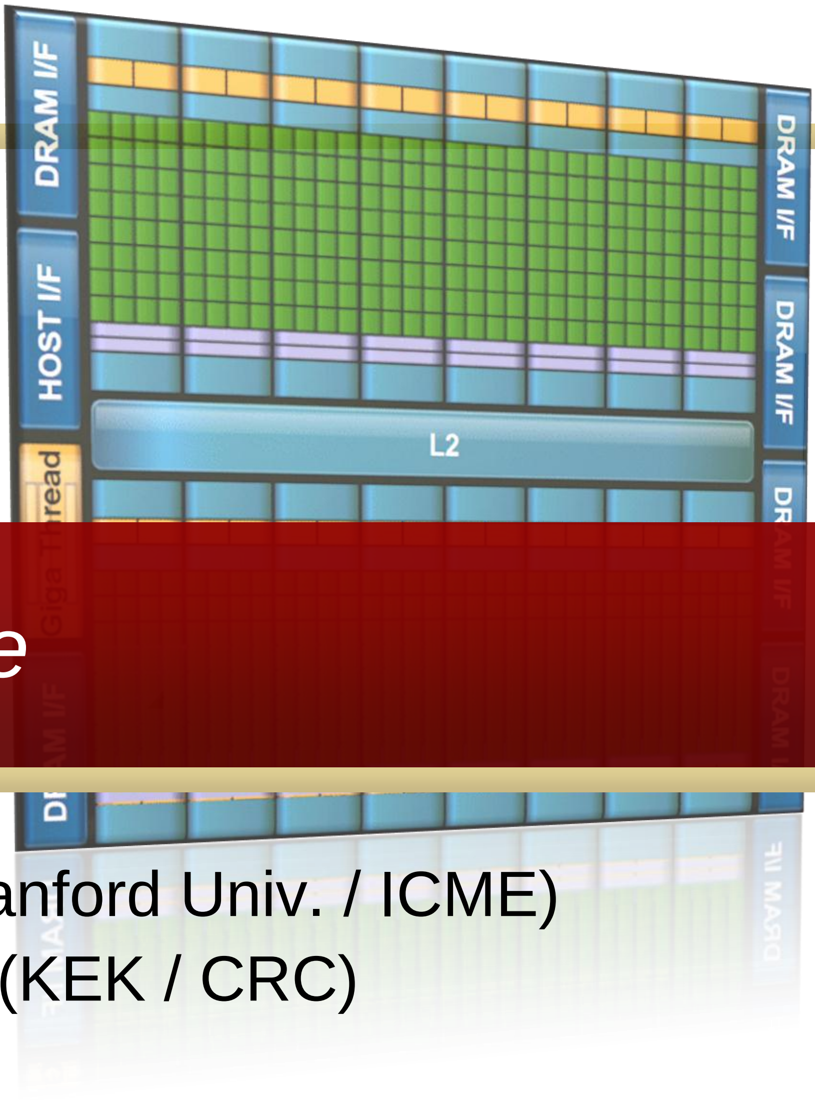


Geant4 on GPU prototype

Nicholas Henderson (Stanford Univ. / ICME)
Koichi Murakami (KEK / CRC)

Stanford ICME, SLAC, G4-Japan Collaboration
supported by NVIDIA

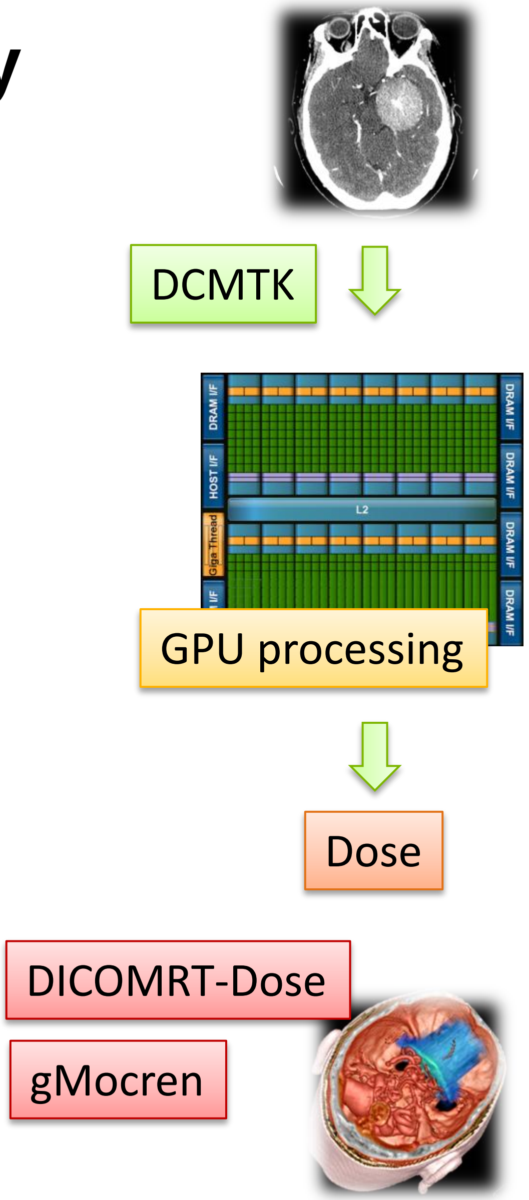
Outline

- project goal
- CUDA basics
- algorithm and implementation
- prototype and performance

Project Goal

Dose calculation for radiation therapy

- GPU-powered
  - parallel processing with *CUDA*
  - boost-up calculation speed
- voxel geometry
  - including DICOM interface
  - material : water with variable densities
- limited Geant4 EM physic processes
  - electron/positron/gamma
  - medical energy rage (< 10-100 MeV)
- scoring dose in each voxel



CUDA basics I

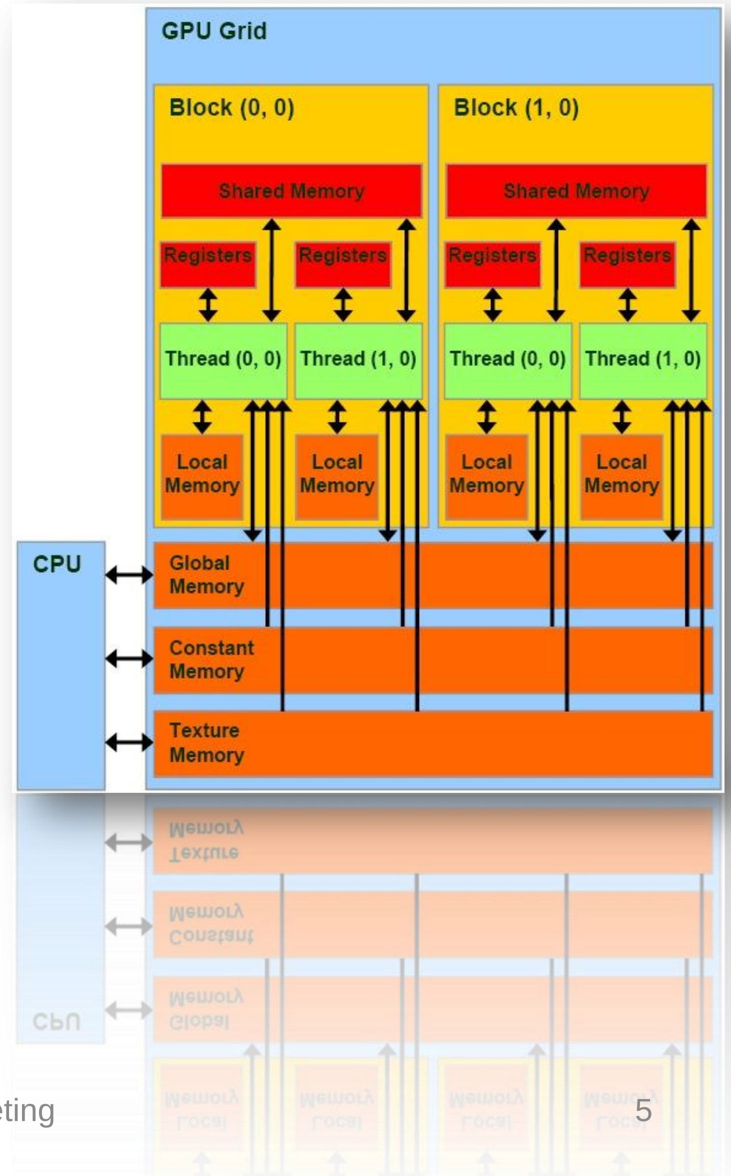
- "***SIMD***" architecture : ***S**ingle **I**nstruction, **M**ultiple **D**ata*
  - CUDA is a data parallel language
  - wants to run same instruction on multiple pieces of data
  - *Think parallel!*

- Coalesced memory access
  - NVIDIA GPUs read from memory in 128 byte blocks
  - To maximize memory throughput, we want a single read to satisfy as many threads as possible
  - We use a "struct of arrays" data structure to maximize opportunities for coalesced memory reads

CUDA Basics II

- Memory hierarchy
  - CUDA provides access to several device memory types:
    - global, shared, constant, texture
  - We currently use global memory for all thread and track data and constant memory for parameters
  - We will use shared memory, which is on-chip, at a later phase in the project

- Race conditions
  - arise when multiple CUDA threads attempt to write to same location in global memory
  - we avoid race conditions (or using atomic operations) by maintaining independent track and dose stacks for each thread



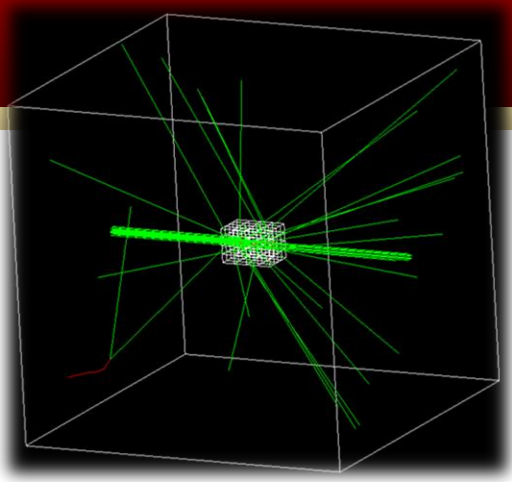
Parallelization strategy

- Each GPU thread processes a single track until the track exits the geometry
  - GPU runs ≈ 32k CUDA threads under the current configuration
- Each thread has two stacks :
  - one for storing secondary particles
  - one for recording the energy dose in a voxel
- After a number of steps :
  - energy dose in the stack is moved to main dose array
  - secondary stacks may be redistributed for performance

G4CU Basics

- Each **thread** stores data for:
  - thread state {running, stopped}
  - PIL(-*left*) for the step
  - the limiting physics process for the step
- Each **thread** processes a **track**, which stores data for:
  - particle spices
  - position
  - direction
  - energy
- Other data associated with each thread:
  - random number generator state, primary generation state, track stack, dose stack, physical process data

Geometry



- Focused on *voxel navigation*
  - taking advantage of GPU power
- Implementation currently handles a single box with uniform discretizations for each dimension
  - planning for a hierarchical voxel model to allow higher resolution in certain regions
- The material of each voxel is water with different density
  - cross section, energy loss, etc are proportional to density
  - not necessary to preparing thousands of tables

Physics processes

- particles : electron, positron, gamma
- energy range : < 10-100 MeV
- material: water (and air) with variable densities
- processes:

  electron / positron
    - energy loss (ionization, bremsstrahlung)
    - multiple scattering (different models will be tried)
    - positron annihilation

  gamma
    - Compton scattering
    - photo electric effect
    - gamma conversion
- physics tables
  - cross section, dE/dx, range, etc are retrieved from Geant4
  - prepared for "*standard*" water

Major algorithm phases

1. initialization
   - allocate memory, initialize RNG (Random Number Generator)

2. main loop
   - **always** take a step
   - sometimes check termination conditions
   - sometimes generate primary particles
   - sometimes pop a secondary particle from track stack
   - sometimes balance track stacks
   - sometimes distribute dose stacks to main dose array

3. clean up
   - output dose
   - free all memory

Algorithm pieces

- **termination check:**
  - algorithm stops if all tracks are stopped, all stacks are empty, and all primary budgets are exhausted
- **primary generation:**
  - The generation procedure generates primaries and pushes them onto stack until stack size reaches a fill_level
  - If the stack size for a given thread is equal to or larger than fill_level, then nothing is done
- **stack pop:**
  - if possible pop a track from the stack and compute initial PILs for all processes

A single step

1. select process with the shortest PIL

2. apply all continuous processes
   - processes have access to the stack

3. decrease PILs of all processes by step path length

4. apply discrete method of the limiting process
   - process has access to the stack
   - resample PIL for the limiting process
   - update transportation PIL if particle properties have changed

Dose accumulation

Notes:

- should avoid race condition due to memory access
- each thread has a dose stack to record deposit energy accumulation
- a thread will push "*the sum of energy deposition in a voxel*" to stack when a track to a different voxel
- periodically, dose stacks will fill up and need to be distributed to the main dose array

Dose distribution procedure:

1. sort all dose stacks together by voxel index
2. reduce (sum-up) by voxel index
3. store the result in main dose array
    - no race conditions because voxel indices are now unique

Example configuration

GPU:

Tesla C2070 (Fermi)

448 cores, 1.15 GHz, 6GB GDDR5 (ECC)

SDK:

CUDA 4.2 (5-RC) : CURAND, Thrust, SCons (CMake)

Application

- generate 200k primaries
- 128 blocks, with 256 threads each
  - total 32,768 CUDA threads
- run takes about 1.2 GB on device (mostly voxel arrays)
- generate and pop every 25 steps
- check termination conditions every 1,000 steps

Current fake "physics" processes

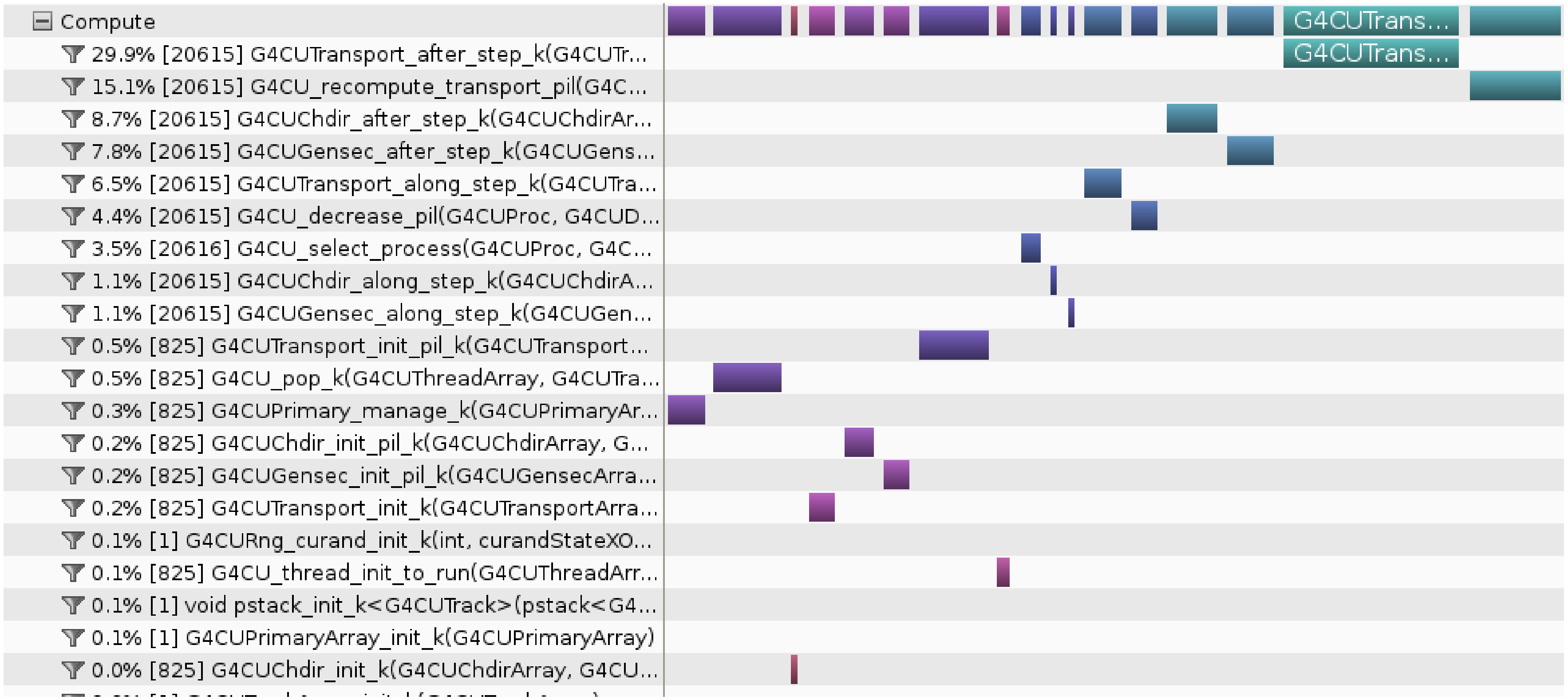
- Compute PIL with:

  ```
  -logf(curand_uniform(&data.rng.state[id]));
  ```

- **Chdir:** perturbs the direction of the particle
- **Gensec:** generates a secondary particle with less energy

NVIDA Visual Profile (nvvp)

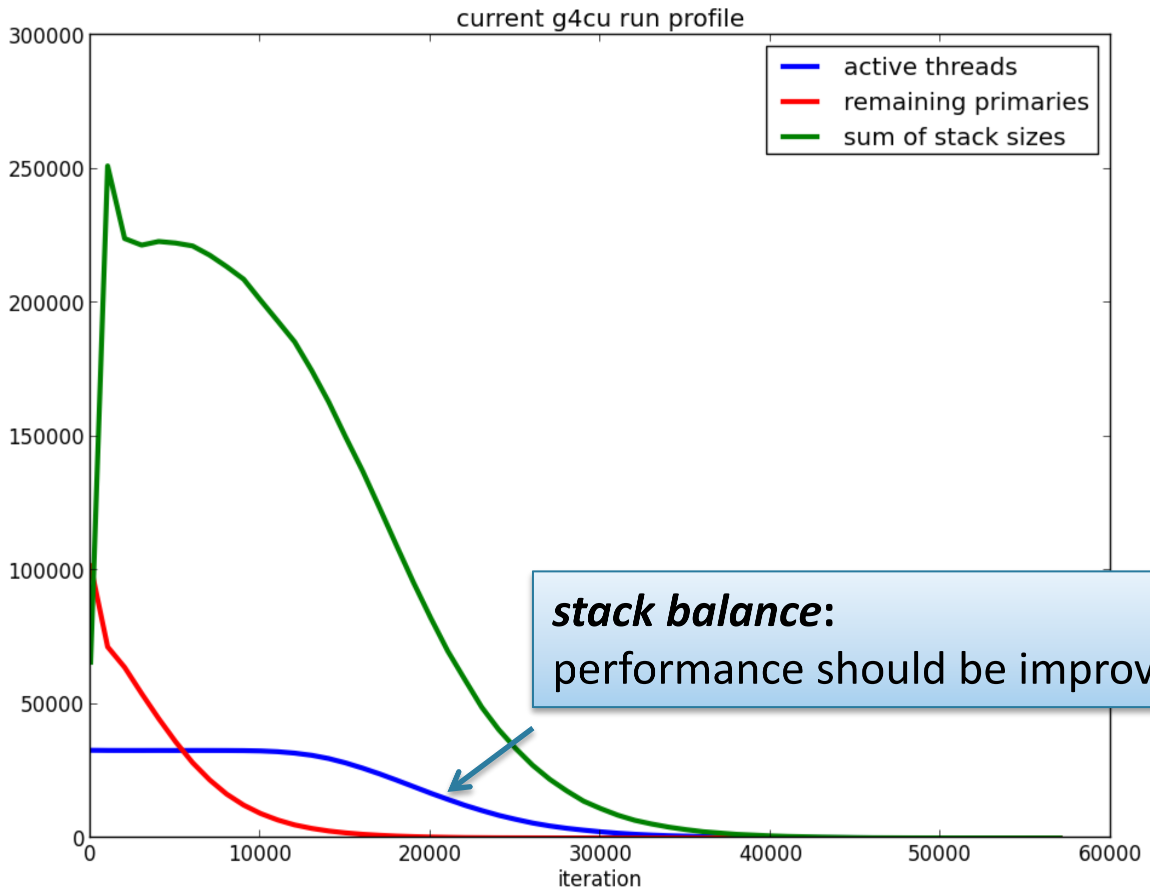
Output from *nvvp* zoomed to a single iteration



```
⊟ Compute
  29.9% [20615] G4CUTransport_after_step_k(G4CUTr...
  15.1% [20615] G4CU_recompute_transport_pil(G4C...
  8.7% [20615] G4CUChdir_after_step_k(G4CUChdirAr...
  7.8% [20615] G4CUGensec_after_step_k(G4CUGens...
  6.5% [20615] G4CUTransport_along_step_k(G4CUTra...
  4.4% [20615] G4CU_decrease_pil(G4CUProc, G4CUD...
  3.5% [20616] G4CU_select_process(G4CUProc, G4C...
  1.1% [20615] G4CUChdir_along_step_k(G4CUChdirA...
  1.1% [20615] G4CUGensec_along_step_k(G4CUGen...
  0.5% [825] G4CUTransport_init_pil_k(G4CUTransport...
  0.5% [825] G4CU_pop_k(G4CUThreadArray, G4CUTra...
  0.3% [825] G4CUPrimary_manage_k(G4CUPrimaryAr...
  0.2% [825] G4CUChdir_init_pil_k(G4CUChdirArray, G...
  0.2% [825] G4CUGensec_init_pil_k(G4CUGensecArra...
  0.2% [825] G4CUTransport_init_k(G4CUTransportArra...
  0.1% [1] G4CURng_curand_init_k(int, curandStateXO...
  0.1% [825] G4CU_thread_init_to_run(G4CUThreadArr...
  0.1% [1] void pstack_init_k<G4CUTrack>(pstack<G4...
  0.1% [1] G4CUPrimaryArray_init_k(G4CUPrimaryArray)
  0.0% [825] G4CUChdir_init_k(G4CUChdirArray, G4CU...
```

Run profile

- Run profile with current fake physics processes and no stack balancing



stack balance:
performance should be improved.

Summary

- Collaborative activity on Geant4-GPU between
  - Stanford ICME, SLAC, and G4-Japan (KEK), supported by NVIDIA

- Focused on medical application
  - dose calculation in voxel domain
  - Geant4 EM physics processes

- GPU prototype
  - parallel tracking on GPU thread
  - multiple data structure for parallel processing
  - efficient stack management

- Working on
  - porting physics processes from Geant4
  - optimization