pCT Image Reconstruction – A Huge Linear Problem

Ákos Sudár [1,2]
Gergely Gábor Barnaföldi [1],
Mónika Varga-Kőfaragó [1]
and Légrády Dávid [2]

[1] Wigner Research Centre for Physics
[2] Budapest University of Technology and Economics

GPU Day 2022 – Massive parallel computing
for science and industrial application
Budapest, 20. June 2022

Motivation and role of proton imaging

- Nowadays the importance of the proton therapy is increasing
  $\Rightarrow$ more and more motivation to improve the technology
- The use of proton CT images is a promising direction
  $\Rightarrow$ lower inaccuracy in RSP measurement
  $\Rightarrow$ decreased safety zone around the tumour
- A pCT image measures the relative stopping power (RSP) distribution of the patient

Bergen pCT collaboration

- Goal: reach the clinical research with a pCT prototype
- Apply monolithic active pixel sensors (MAPS)
- Use pencil beam for imaging
- Measure 10^6 proton / second
- Reach $<$ 1 % RSP error

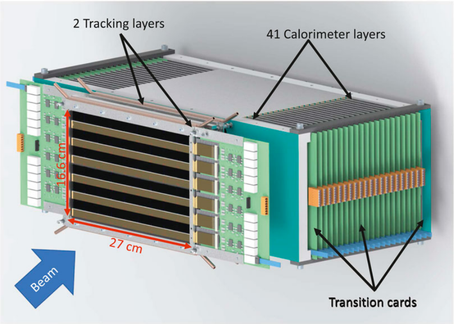


Image reconstruction – a large linear problem

The image reconstruction is a large and sparse linear problem:

$$\mathbf{y} = \mathbf{A}\,\mathbf{x}\,,$$

where:

- $\mathbf{y}$ is the measured data
- $\mathbf{x}$ is the vector of voxels
- $\mathbf{A}$ is the system matrix, contains the intaraction coefficients
  - practically the path length of protons in the voxel
  - can have 10^{12} non zero element – about 12 Tbyte
    $\Rightarrow$ on the fly calculation of the element instead of store them
  - matrix element become a function: $A_{i,j} \;\Rightarrow\; A(i,j)$

Hardware

Hardware:

- 4 piece of Nvidia 1080Ti
- computer capability: 6.1
- CUDA version: 11.2

Reconstructed Derenzo phantom

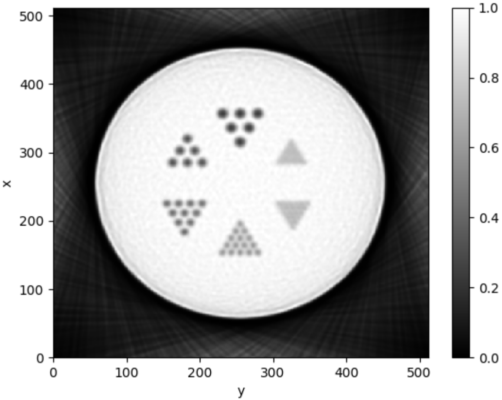
Reconstructed Derenzo phantom after 250 iterations:

Without errors:

- Exactly restored image

With errors:

- Reasonably good spatial resolution
- Point spread function FWHM = 4.3 mm
- Acceptable RSP accuracy



Development 1 – Optimize the memory access

First implementation

- Variables in GPU memory
- Access during use
  ⇒ not coalesced access
  ⇒ access each data multiplied times

Optimized memory access

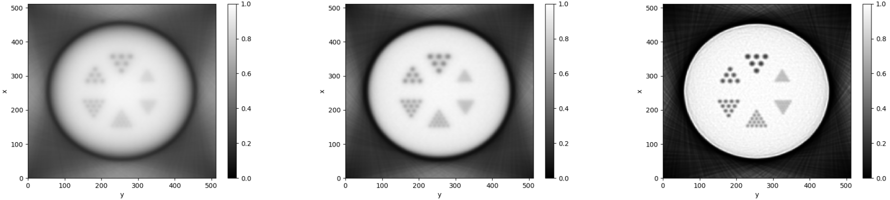
- Variables in GPU memory
- Copy to shared memory
  ⇒ coalesced access
  ⇒ access each data once per thread block

Summary

The steps of the optimization:

Phase	**Exp. time (h)**	**State**
Dirty but working	$\sim$ 100 000	Finished
Coalesced memory access	6494	In progress
Minimized calculations	269	Planned
Optimized algorithm	2.7	Planned

Thank you for your attention!



I would like to thank the support of the Hungarian National Research, Development and Innovation Office (NKFIH) grants under the contract numbers OTKA K135515 and 2019-2.1.6-NEMZ_KI-2019-00011. Computational resources were provided by the Wigner Scientific Computing Laboratry.

Backup slides

Backup slides

Image reconstruction – Richardson – Lucy algorithm

- Originally introduced for astrophysics application
- It is a fixed point iteration for large and sparse linear problems
- Initialization: arbitrary positive vector
- Init: unit vector or precalculated approximate solution

The formula for the i^{th} element of the next image vector:

$$x_i^{k+1} = x_i^k \frac{1}{\sum\limits_j A_{i,j}} \sum_j \frac{y_j}{\sum\limits_l A_{l,j} x_l^k} A_{i,j} \ ,$$

where k is the number of iteration. 20-300 iteration is typical.

Parallelization & avoidance of multiply calculations

Update the i^{th} voxel in the k^{th} iteration:

$$x_i^{k+1} = x_i^k \frac{1}{\sum\limits_j A(i,j)} \sum_j \frac{y_j}{\sum\limits_l A(l,j) x_l^k} A(i,j)$$

$$\Downarrow$$

$$x_i^{k+1} = x_i^k N_i \sum_j \frac{y_j}{\sum\limits_l A(l,j) x_l^k} A(i,j)$$

Pre-calculate the normalization of the i^{th} voxel:

$$N_i = \frac{1}{\sum\limits_j A(i,j)}$$

Parallelization & avoidance of multiply calculations

Update the i^{th} voxel in the k^{th} iteration:

$$x_i^{k+1} = x_i^k N_i \sum_j \frac{y_j}{\sum_l A(l,j) x_l^k} A(i,j)$$

$$\Downarrow$$

$$x_i^{k+1} = x_i^k N_i R_i^k$$

$R_i = 0$. For i^{th} voxel and j^{th} proton history:

$$R_i^k \mathrel{+}= \frac{y_j}{\sum_l A(l,j) x_l^k} A(i,j)$$

Parallelization & avoidance of multiply calculations

Update the i^{th} voxel in the k^{th} iteration:

$$x_i^{k+1} = x_i^k N_i R_i^k$$

First: Calculate the Hadamard ratio (once per iteration):

$$H_j^k = \frac{y_j}{\sum_l A(l,j) x_l^k}$$

Second: $R_i = 0$. For i^{th} voxel and j^{th} proton history:

$$R_i^k \mathrel{+}= H_j^k A(i,j)$$

GPU algorithm

Algorithm 1 GPU algorithm

GPU: calculate voxel normalization
for needed number of iterations **do**
    while end of proton histories **do**
        CPU: read certain amount of proton histories
        GPU: calculate Hadamard ratio:

- parallel calculation of proton histories
- serial calculation of voxel interactions

        GPU: calculate voxel contribution

- serial calculation of proton histories
- parallel calculation of voxel interactions

        GPU: Update the image vector
    end while
end for
CPU: Save the image vector