Accelerating C++ applications in Medical Physics

Pedro M. M. Correia

PhD Student
Physics Department
University of Aveiro, Portugal
e-mail: pmcorreia@ua.pt

Inverted CERN School of Computing, 29 February – 2 March 2016
Outline

- Positron Emission Tomography (PET)
  - What is a PET scanner?
  - Detector simulation, data acquisition/analysis and image reconstruction in PET (GATE simulations and data analysis)

- Acceleration tools:
  - Intel® Threading Building Blocks
  - OpenMP API (Open Multi-Processing Application Programming Interface)
  - NVIDIA CUDA®

- Case Studies:
  1. Parallelism in GATE
  2. Image Reconstruction: OpenMP vs TBB
  3. MLEM Acceleration

- Conclusions
Positron Emission Tomography (PET)
Positron Emission Tomography (PET)

- Radioactive nucleus decays in a $\beta^+$ reaction.
- $\beta^+$ annihilates -> two antiparallel 511 keV photons emitted.

In *Basics of PET Imaging Physics, Chemistry, and Regulations*, Gopal B. Saha
Positron Emission Tomography (PET)

- Detection of the two photons in the same time window is called a **coincidence** event, and the line is called **LOR** (line of response).

- A patient is injected with a radiotracer (usually FDG, a radioactive replacement of the deoxyglucose) that accumulates in a region of interest.

- The detection of coincident events allows the image reconstruction.
PET Scanner

Applications

Functional clinical studies

- Oncology
- Cardiology
- Neurology

Pre-clinical research

- New drugs
- Diseases research

From *Emission Tomography: The Fundamentals of PET and SPECT.*

Image from Jens Maus (http://jens-maus.de/)
Data Acquisition

Gama photon 511 keV

Optical photon (~eV)

Scintillator (LYSO)

Data acquisition

Discrimination and amplification

Photomultiplier
PET Simulations

- Simulations play an important role in Medical Research
  - Develop and optimize new scanners and techniques
    - PET-CT, PET-MRI, PET-CT-MRI, SPECT-CT, Optical Imaging, etc.
  - Discover of new drugs
    - Biomarkers
  - Study of diseases and new treatments.
    - Cancer, Alzheimer's
    - Radiotherapy and Hadrontherapy
GATE

- **GATE (Geant4 Application for Tomographic Emission)**\(^1,\)\(^2\)
  - Monte Carlo application
  - Allows the use of simplified macros as primary input mechanism (no C++ knowledge is needed for most of the applications);

---

\(^{1}\) Jan, S. et al., “*Gate: a simulation toolkit for PET and SPECT*”

\(^{2}\) Santina, G. et al., “*Evolution of the GATE project: new results and developments*”
GATE Architecture

1. Defining the geometry

2. Defining the source/phantom

3. Start acquisition/simulation

4. Analysis and image reconstruction
GATE simulation

- Geant4 engine is used for generate and tracking particles

- In a typical simulation:
  - Source with 300 MBq $^{18}\text{F}-\text{FDG}$ generates $\sim 10^{11}$ decays during a 30 min scan (plus secondary particles)
  - Each event (decay) is independent – Monte Carlo simulation
  - Very time consuming to track all this particles

- Parallelization first approach:
  - Spit job into smaller jobs (time slices) in a grid

```
1 Job – 30 Min
   \rightarrow
Slice 1 - 1 min

(...)

Slice N - 1 min
```
Image Reconstruction algorithms

- **Analytic methods** (fast, simpler and easier to implement). E.g. Retroprojections.

- **Iterative methods** (slower, more complex but usually with better performance)

From *Emission Tomography: The fundamentals of PET and SPECT.*
Image Reconstruction Methods

- The “inverse” problem of the acquisition. Why?

- **Analytical:**
  - Involves the reconstruction of an image from its X-Ray transform
  - Deterministic problem, usually “ignoring” real data.
  - Efficient and non-iterative algorithms

- **Iterative:**
  - Start with a guessing image
  - Finite iterations over images
  - Requires heavier calculations
  - Suitable for more complex problems

  - Backprojection
  - Filtered Backprojection
  - MLEM
  - OSEM
Analytical reconstruction

- **Uses X-Ray transforms:**
  1. Returns all the possible line integrals of an image $f(x,y)$.

2D example:
- The X-Ray transform is the operation $f(x,y) \rightarrow p(x_r, \phi)$
- $p(x_r, \phi)$ is the 1D projection of $f(x,y)$ for a given angle $\phi$

From *Emission Tomography: The fundamentals of PET and SPECT*
Analytical reconstruction

- **Uses X-Ray transforms:**
  - **2. Central slice theorem**
    - Gives relation between 2-D Fourier transform of image and 1-D Fourier transform of its projections along the detector axis

\[ P(\nu_{xr}, \phi) = F(\nu_x, \nu_y) \bigg|_{\nu_{yr}=0} \]

- 1-D FT for projections along axis at angle \( \phi \)
- 2-D FT of the image + central slice

![Diagram depicting analytical reconstruction process](image-url)
Analytical reconstruction

- **Uses X-Ray transforms:**
  - **3. 2-D FBP algorithm**
    - Compute 1-D FT of the projections along the detector axes
    - Apply:
      - ‘Ramp-filter’ in the frequency space (1-D Convolution)
      - 1-D iFT to obtain filtered projections
      - Back-projection operator to obtain the image

\[
f(x, y) = \int_{0}^{\infty} \int_{-\infty}^{\infty} |u_{xr}| P(v_{xr}, \phi) e^{i2\pi v_{xr} u_{xr}} dv_{xr} d\phi
\]

- Parallelization can be obtained, for example:
  - Projections for different \( \phi \) may be calculated in different processors
Iterative Reconstruction

- Different iterative methods are available. Ex: ML-EM, OSEM, …

- **Maximum Likelihood - Expectation Maximization (ML-EM)**
  - First introduced for image reconstruction in 1982 by Shepp and Vardi, remains the **basis** algorithm for iterative statistical image reconstruction
  - It leads to the iterative equation and chart[1]

\[
\hat{f}_j^{(n+1)} = \frac{\hat{f}_j^{(n)}}{\sum_{i'} H_{i'j}} \sum_i H_{ij} \sum_k \frac{p_i}{H_{ik} \hat{f}_k^{(n)}}
\]

Image at iteration n+1

Acceleration tools
Acceleration tools

- Many tools and approaches have been developed in the last years to extract as much performance as the recent computers can give.

- Some examples:
  - Multi-thread (sharing memory): e.g. OpenMP, TBB, CUDA
  - Multi-CPU (splitting memory): e.g. MPI
  - Intrinsic parallelism: split the work in smaller parts

- Monte Carlo simulations are well suitable for intrinsic parallelism[1].

Data parallelism vs Task Parallelism

- We can think about parallelism in two ways:
  - **Task parallelism**: Simultaneous execution of different tasks on the same or different data
  - **Data parallelism**: Simultaneous execution of the same task/function (single instruction, multiple data – SIMD) for various elements on a ensemble

- The use of one or the other approach depends on the user application, usually the use both is the best option
OpenMP

- First released in 1997

- Designed to ensure an ordered access of different threads to shared data and to be a standard notation among different SMP (Symmetric multiprocessing) architectures.
- Supports Fortran, C and C++.
- It’s implemented in many commercial and Open Source compilers.
- It’s a set of compiler directives, library routines and environment variables as an extension of C, C++ and Fortran standard compilers.
- For simple applications, only few code lines may be needed.
OpenMP

- Uses the Fork-join\cite{1} model of parallel execution

![Diagram showing Fork-join model]

- For-loop example C++:

```cpp
// Typical C++
for (int i = 0; i < 8; ++i) {
    do_some_task(i);
}

// C++ with OpenMP
#include <omp.h>
omp_set_num_threads(8);
#pragma omp parallel for
for (int i = 0; i < 8; ++i) {
    do_some_task(i);
}
```

\cite{1}M. E. Conway. A multiprocessor system design. In Proceedings, November 12-14 1963
Intel® Threading Building Blocks

- "Is a popular software C++ template library that simplifies the development of software applications running in parallel" from https://www.threadingbuildingblocks.org/faq

- Unlike OpenMP, TBB makes use of the typical programming style of C++

- It is focused for tasks instead of threads

---

Task queue

- Thread 1
- (...)
- Thread N
Accelerating C++ Applications in Medical Physics

Intel® Threading Building Blocks

For-loop example C++:

```cpp
#include "tbb/tbb.h"
using namespace tbb;

void Application(size_t size) {
  parallel_for(size_t(0), size, size_t(1), [=](size_t i) {
    do_some_task(i);
  });
}

int main(){
  const size_t size = 8;
  Application(size);
  return 0;
}
```
CUDA®

- **Compute Unified Device Architecture**, introduced in November 2006 by NVIDIA


- It allows the use of GPUs to solve complex parallelization problems that general CPUs have more difficulty/need more time to handle.

- To make use of it, the installation of a software environment is needed. Its possible to develop applications using different programming languages (C, C++, Fortran, Java, etc).

- Its being widely used in scientific applications nowadays.
CUDA®

- Three major steps:
  - Copy the data for processing from CPU memory to GPU memory
  - Execute the desired processing
  - Copy results back to the CPU memory

- The main concept
  - The used defines a function, called kernel, and each kernel, when called, is executed in \textbf{N threads} in parallel
  - Each thread has \textbf{an unique ID}
  - Threads can be grouped in \textbf{blocks}, and blocks in \textbf{grids} – the dimensions of each depends on the type of data for processing

Case Study 1

Parallelism in GATE
Parallelism in GATE

- Two possibilities for simulations acceleration:

  - **Time split:**
    - Events are dependent between them.
    - Jobs are separated in the time domain into smaller jobs and distributed among a queuing system.
    - Better for imaging applications, due to time dependant effects

  - **Events split:**
    - Events are independent
    - Each job will only simulate a fraction of the total number of events
    - More suitable for Dose Applications (Radiotherapy, Hadrontherapy) because no dependence between particles is demanding
Parallelism in GATE

- **Using multithreading capabilities with CUDA:**
  - Only **highly-demanding** parts of the simulation go to CUDA kernels - hybrid simulation CPU+GPU

- **Phantom part uses CUDA**
  - One particle per thread
  - Specific kernels for physics effects

- **Detector part uses the CPU**
  - Time dependence is vital to simulate electronic chain, data acquisition and reconstruct images with accuracy

- **Example:** Hybrid-GATE project, funded for 36 months by the French National Research Agency, to accelerate GATE simulations using CPU/GPU capabilities
Parallelism in GATE

- Geant4 code has been moved to GPU (random number generator, photon physics effects, etc)[1]

---

GATE CPU+GPU

Voxelized phantom

GATE (CPU)
10^6 particles ~90 s

GATE (CPU+GPU)
10^6 particles <2 s

Case Study 2

Image Reconstruction: OpenMP vs TBB
Image Reconstruction
OpenMP vs TBB

- Image reconstruction is also usually a time consuming task.

- **LM OSEM**, an iterative algorithm for 3D image reconstruction:
  - PET events (LORs) are split into $s$ equally spaced subsets
  - For each subset $l \in 0, \ldots, s-1$, is calculated $f_{l+1}$:

  $$f_{l+1} = f_l c_l; \quad c_l = \frac{1}{A_N} \sum_{i \in S_l} (A_i)^t \frac{1}{A_i f_i}.$$

  - Where $f \in \mathbb{R}^n$ is a 3D image in vector form with dimensions $n = (X \times Y \times Z)$.
  - $A \in \mathbb{R}^{m \times n}$ and the element $a_{ik}$ of the row $A_i$ is the length of the LOR correspondent to the event $i$ and the voxel $k$, calculated with Siddon’s algorithm$^{[1,2]}$.

---


Image Reconstruction
OpenMP vs TBB

- **LM OSEM algorithm has 3 nested loops:**
  - 1 outer loop over all subsets
  - 2 inner loops, one for the summation, another for the iterations

```c
for (int l = 0; l < subsets; l++) {
   /* read subset */

   /* compute c_l */
   #pragma omp parallel
   {
      #pragma omp for schedule(static)
      for (int i = 0; i < subset_size; i++) {
         ...
      }
   } /* end of parallel region */

   /* compute f_l+1 */
   #pragma omp parallel for schedule(static)
   for (int k = 0; k < image_size; k++) {
      if (sens[k] > 0.0 && c_l[k] > 0.0)
         f[k] = f[k] * c_l[k] / sens[k];
   }
```

Using OpenMP

Race condition might happen here
Image Reconstruction

OpenMP vs TBB

- **LM OSEM algorithm has 3 nested loops:**
  - 1 outer loop over all subsets
  - 2 inner loops, one for the summation, another for the iterations

- For TBB, code modifications are needed

```c++
class ImageUpdate {
    double *const f, *const c_l;
    double *const sens,

public:
    ImageUpdate(double *f, double *sens, double *c_l) :
        f(f), sens(sens), c_l(c_l) {}

    void operator() (const blocked_range<int>& r) const {
        for (int k = r.begin(); k != r.end(); k++) {
            if (sens[k] > 0.0 && c_l[k] > 0.0)
                f[k] *= c_l[k] / sens[k];
        }
    }
};
```

Using TBB
Image Reconstruction
OpenMP vs TBB

- LM OSEM algorithm has 3 nested loops:
  - 1 outer loop over all subsets
  - 2 inner loops, one for the summation, another for the iterations

- For TBB, code modifications are needed

```cpp
for (int l = 0; l < subsets; l++) {
    /* read subset */

    /* compute c_l */
    parallel_for(
        blocked_range<int>(0, subset_size, GRAIN_SIZE),
        SubsetComputation(f, c_l, event_buffer, precision));

    /* compute f_{l+1} */
    parallel_for(
        blocked_range<int>(0, image_size, GRAIN_SIZE),
        ImageUpdate(f, sens, c_l));
}
```
Image Reconstruction
OpenMP vs TBB

- **Implementation results:**

1. Preventing race conditions (using mutexes or critical sections), **OpenMP** has shown **better** performance over **TBB**.

2. Using **OpenMP** requires very little program redesign, contrary to **TBB**.

3. **TBB** is more suitable for the design of new applications from the scratch, while **OpenMP** is preferable to redesign already developed code.

Running machine: dual quad-core (AMD Opteron 2352, 2.1GHz with
Case Study 3

MLEM Acceleration
MLEM Acceleration

- The most of the time spent in projection and backprojection operations between the detected LORs and the image voxels.

- Acceleration of the method has been achieved in recent years, using single GPUs.

- Image reconstructions for real-time applications in hospitals demands even higher speed-up of calculations.

- Multi-GPUs systems are being used\(^1\) but the communication between GPUs is a bottleneck.

- GeForce GTX 480 and GTX 285 were tested

\(^1\) Distributed MLEM: An Iterative Tomographic Image Reconstruction Algorithm for Distributed Memory Architectures 
Craig S. Levin et al, IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 32, NO. 5, MAY 2013
MLEM Acceleration

- In this work, the common MLEM (DG) algorithm is described as a special case of a general optimization problem.

- A new MLEM (DMLEM) is derived by maximizing the same likelihood function as the common MLEM, but adapted to a multiGPU system.

- The new algorithms perform several iterations in sub-problems of the original problem, but minimizing the same objective function.
MLEM Acceleration

- Overall difference between images obtained with the two methods is negligible after the initial iterations.

- Linear speedup with increase of GPUs is obtained in both methods for small number of GPUs, but saturation of DMLEM occurs later because of the reduced communication between independent nodes.
Conclusions

- In Medical Physics, several applications have taken advantage of the rapidly increase of the computation resources that are available, specially for:
  - Simulation of the operation of existent and new scanners.
  - Image reconstructions both for scientific or preclinical research and in real time clinical practice.

- New software tools have been used to accelerate these tasks:
  - OpenMP and TBB, using the multi-CPU capabilities.
  - Nvidia CUDA, taking advantage of the power of graphic cards commonly available.

- The use of one tool instead of the other will depend mainly on:
  - The desired application (new or renewed).
  - The available resources (time, funds, hardware).
  - The will of the person in charge.