Graphic Processors in Computational Applications

List of Projects

dr inż. Krzysztof Kaczmarski

2024

Materiały sponsorowane przez:

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Fundusze Europejskie
Wiedza Edukacja Rozwój

Rzeczpospolita Polska

Politechnika Warszawska

Unia Europejska
Europejski Fundusz Społeczny

This set of slides is entirely new. Although, I did my best you may find mistakes or typos. I reward 1 point extra for every case reported if you are the first one to spot it.

Grading rules I

Obligatory Requirements

Points

Each student prepares two projects (two levels of difficulty) and must collect at least 50 points out of 100.

Time measurement

Each project must measure separated time including:

- data generating or reading (if applicable)
- data CPU–GPU copying
- important algorithm stages

Programs with graphic animations should measure frames per second (FPS) metric.

Grading rules II

Obligatory Requirements

GPU, CPU Comparison

Each project must contain sequential (CPU) and parallel (GPU) versions of the solution and should be able to perform execution time comparison. CPU version of an algorithm not necessarily has to be implemented by a student. For example, if we consider *quicksort*, then for CPU version one can use standard C qsort() function.

This requirement may be omitted after consultation and acceptance by the teacher in special cases only.

Points vs grades

0-49: **2**, 50-59: **3**, 60-69: **3.5**, 70-79: **4**, 80-89: **4.5**, 90-100: **5**

Grading rules III

Obligatory Requirements

Penalty points

- delays – 10% for every week of delay
- execution problems – up to 50%
- wrong solution – 10% each wrong construct (for example Array of Structures used instead of Structure of Arrays)
- missing functionality – up to 100%

Disclaimer

If a student cannot explain the project contents or cannot present the algorithm used in a convincing way the project is rejected as it is.

You must report progress every two weeks.

Grading rules IV

Obligatory Requirements

Example of penalty points

-10% : processor occupancy not achieved or too few threads running

-10% : memory allocation or deallocation problems

-10% : AoS if SoA is possible

-5% : shared memory conflicts

-5% : ugly code, no comments, mess in files

-5% : no makefile (cmake is ok)

Classes Organization I

Subsequent weeks:

1	Introduction, Choose platform, Check environment, Remote Desktop, Run samples
2	Training task 1 (textual console) – presence – collect 3p
3	Training task 2 (graphical output) – presence – collect 3p
4	Training task 3 (api) – presence – collect 3p
5-9	Project 1
10-14	Project 2

Remember about delivering a short implementation plan for your project in the half-time of its realization.

Classes Organization II

Platforms

Windows	Vistual Studio (Nsight plugin), Remote Desktop
Linux	Nsight, ssh

Linux Servers

gpunode1	1 x K20, for BSc and MSc students anytime
gpunode2	1 x K40, for BSc and MSc students anytime
gpunode3	1 x T4, temporarily for industrial project
eden cluster	32 x A100, 8 x H100, 4 x P100, research only

List of Projects

Easy projects (40p)

1.1 Electrons and protons
1.2 Raycasting of spheres
1.3 Raycasting of triangles
1.4 Shoal of fish animation
1.5 Sudoku solver
1.6 Cosmic radiation scanner
1.7 Gravitation box
1.8 English Peg Solitaire solver
1.9 Clustering using k-means algorithm.

Moderate projects (60p)

2.1 Raycasting of a CSG tree of spheres
2.2 BFS for large graphs
2.3 Fast FL/RL compression
2.4 Monte-Carlo Tree Search for Checkers
2.5 Monte-Carlo Tree Search for Go
2.6 Hamming one
2.7 Levenshtein distance

1. Easy projects (40p)

1.1 Electrons and protons

Easy projects (40p)

Goal

Visualize of an electrostatic field with moving particles: electrons (-) and protons (+).

Details

Input: Random set of particles placed in 2d space (at least 10k)

Output: Graphics displayed in a window, user may stop movement (at least 30fps)

Initial position and speed vector of the particles may be random. Particles should move according to the electrostatic force and bounce off the window frame. Electrostatic field value should be presented as a color intensity of the pixels in the in window, for example red(+) and blue(-). Particles may be drawn as black dots.

Remarks

Please remember about SoA principle. There are two stages of the program: visualization of the filed (single pixel one cuda thread) and calculation of the movements in the filed (single particle one cuda thread). The second one may use results from the first one. Shared memory is important in this project.
Extra points (5p) for automatic scaling of the resolution of the view when a window frame is resized.

1.2 Raycasting of spheres

Easy projects (40p)

Goal

Visualize a set of spheres in 3d space using simple raycasting using Phong reflection model with many light sources.

Details

Input Random set of colored spheres (at least 1k) and color sources of light in 3d space (at least 10)

Output Graphics displayed in a window, user can rotate the scene or lights

Remarks

There are beams sent from the user's eye perpendicularly to the screen, one beam goes through one pixel. Color of the pixel must be calculated when the beam hits the closest sphere (you may create a z-buffer). At the hit point hit angle can be easily calculated having the sphere position and radius.
Phong reflection model is described for example here:
https://en.wikipedia.org/wiki/Phong_reflection_model.
In raycasting viewer direction vector is parallel to Z axis.
Extra points (5p) for automatic scaling of the resolution of the view when a window frame is resized.

1.3 Raycasting of triangles

Easy projects (40p)

Goal

Visualize a set of triangles in 3d space using simple raycasting using Phong reflection model with many light sources.

Details

Input A file with a model of an object composed of triangles (at least 10k) and a file with color sources of light in 3d space (at least 10)

Output Graphics displayed in a window, user can rotate the scene or lights

Remarks

There are beams sent from the user's eye perpendicularly to the screen, one beam goes through one pixel. Color of the pixel must be calculated when the beam hits the closest triangle (you may create a z-buffer). At the hit point hit angle can be easily calculated having the triangle position.
Phong reflection model is described for example here:
https://en.wikipedia.org/wiki/Phong_reflection_model.
In raycasting viewer direction vector is parallel to Z axis.
Extra points (5p) for automatic scaling of the resolution of the view when a window frame is resized.

1.4 Shoal of fish animation

Easy projects (40p)

Goal

Visualize 2D simulation of a large shoal of fish.

Details

Input Random set of fish placed in 2d space (at least 10k)

Output Graphics displayed in a window, user may stop movement (at least 30fps)

For the details of a shoal modeling visit: http://www.red3d.com/cwr/boids/. Single subject should be shown by an arrow and has to move according to the modeled behavior. Visualization in a graphical window. Keys control over the basic model parameters would be nice.

Extra

3D visualization (15p), interaction with the fish like avoiding the mouse pointer (5p), different types of fish (5p)

1.5 Sudoku solver

Easy projects (40p)

Goal

Create a program which will solve a given 9x9 sudoku board.

Details

Input A selected sudoku board

Output A solved board or information that there are no solutions.

For each board there should be a set of new boards generated in parallel. Then each one should be checked for consistency and then the process repeats. Be careful with memory consumption and balance wisely between breadth and depth of the search but also utilize parallel processing. Proper board encoding may significantly reduce memory allocation.

Remarks

There are several stages in a single loop iteration. Parallel threads may be used in different ways at each of them. You should utilize all available SMs (many blocks of threads) and cores (many warps). Fixed number of 81 or 9 threads approach throughout all the process is not a good solution.

Extra

Extra 10p for massive parallel boards solving.

1.6 Cosmic radiation scanner

Easy projects (40p)

Goal

Detect cosmic rays deviation on objects. Visualize object's shape and rays.

Details

Input Download from:
https://pages.mini.pw.edu.pl/~kaczmarskik/gpca/CosmicRadiationScanner.7z
(471MB)

Output Visualization the rays including deviation data and the anticipated shape of the scanned object.

Single cosmic particle enters two sheets of detectors above an object and two detectors below an object. Each detector stores coordinates (x, y) of a collision point. Two detectors from the detector above an object define a line of entering. Two detectors below define a line of leaving. Distance and angle between lines describe probability of an object existence on the particle path.
Difficulty of the project lays in several necessary methods of detecting if there was an interaction or not (angle, distance, both, distribution of angles, etc)
There must be complex interaction created in order to choose different detection models and visualizations.

1.7 Gravitation box

Easy projects (40p)

Goal

Visualize 2D simulation of a large set of circles in a gravitational field (at least 100k).

Details

Input Random set of particles placed in 2d space with random velocity. It would be nice to have initial positions forming some shape. Particles' mass may also be random (in some sensible interval) instead of uniform.

Output Visualization of the particles movement.

Consider collisions between objects and the frame of the window.
(see http://www.cs.cmu.edu/∼baraff/pbm/particles.pdf)
Important note: we do not calculate attraction between particles themselves.

Extra

Extra points (5p) for automatic resizing of the box when a window frame is resized.
Particles' radius should not change when the window is resized.

1.8 English Peg Solitaire solver

Easy projects (40p)

Goal

Implement brute force solver for English version of Peg Solitaire (https://en.wikipedia.org/wiki/Peg_solitaire).

Details

Input Size of board edge (3, 5, 7 etc, 3 is a standard size), position of the initial empty slot

Output Sequence of moves leading to a solution if existing.

The program should utilize maximum number of possible resources of a GPU device. The program should try all possible moves and print out the solution and the path leading to this solution. During the computations it should also output information about number of analyzed boards, number of boards without any move, number of performed moves, etc. No graphical output is required. Utilization of another solver in order to find shorter solution is forbidden. The program should include a testing procedure which will be able to verify and confirm that the path found is correct.

1.9 Clustering using k-means algorithm.

Easy projects (40p)

Goal

Implement parallel version of k-means clustering algorithm.

Details

Input Random set of N points (millions) is n-dimensional space. k – number of clusters

Output Testing results.

http://www.eecs.northwestern.edu/~wkliao/Kmeans/index.html
There are two stages: calculating distances from all points to all centroids (embarrassingly parallel) and finding new centroids. The biggest question here is how to optimally calculate new centroids. Two different methods must be implemented and compared.

Extra

For $n = 3$ visualization of points in space: 10p.
If n is a template parameter: 10p.

2. Moderate projects (60p)

2.1 Raycasting of a CSG tree of spheres

Moderate projects (60p)

Goal

Visualize a scene in 3d space using simple raycasting using Phong reflection model (on lighting see slide 13).

Details

Input A file with a scene built of spheres in a CSG tree (hundreds of nodes).

Output Graphics displayed in a window, user can rotate the scene or lights

Each node of a tree may contain $\cap, \cup$ or $\setminus$ operations. The resulting shape is a result for traversing the tree bottom up and performing the operations on the subsequent levels. Assume one white directional source of light.
https://en.wikipedia.org/wiki/Constructive_solid_geometry
The scene should be stored in the shared memory. Use warps to process the tree in a parallel way.

Extra

Adding more types of basic shapes (cube, cylinder, etc.) 10p.

2.2 BFS for large graphs

Moderate projects (60p)

Goal

Implement Breadth-First-Search algorithm for GPU and run in on a set of large graphs from public data (millions of nodes).

Details

Input A public data set graph

Output Path between two nodes

The general idea of the algorithm uses one thread (or one warp) for one visited node. There is a frontier queue created with the nodes to be visited in the next step. Details of the globally non-blocking implementation of the queue are left for implementation: there are generally two approaches with atomic operations on various levels of threads hierarchy (warps, blocks) and without them using multiple buffers from shared to global memory.
A student should implement at least two different approaches for GPU.
Visualization is not required.

2.3 Fast FL/RL compression

Moderate projects (60p)

Goal

Implement two simple lossless compression and decompression algorithms. Both of them are based on local data analysis.

Details

Input A bitmap image can be a good input data for testing.

Output Compressed sequence and decompressed sequence compared to the initial one.

Fixed-Length: In a given buffer read all data and find minimal number of bits required to store the values by removing the highest zeroed-bits. There is no external vocabulary needed. There may be different output bit-length for every buffer frame. Length of the analyzed frame may be chosen arbitrarily.
Run-Length: Count number of subsequent equal values and store this number in a dedicated array. In another array store this value itself.
https://en.wikipedia.org/wiki/Run-length_encoding
Both algorithms should use parallel threads in an efficient way using inter-warp communication methods.
This program should be tested against data of different length (bigger the better) and characteristics.

2.4 Monte-Carlo Tree Search for Checkers

Moderate projects (60p)

Goal

Simulate checkers game using move prediction performed using Monte-Carlo Tree search method.

Details

Input Checkers board

Output Proposed move (in a loop)

The general idea behind this algorithm is to choose next move without any position valuating functions. There is no strategy necessary for this method. From a given board we generate random plays and then calculate wins/looses ratio in subtrees. This ratio is a basis for next move selection.
Simulate a game computer vs computer and allow game with a human.

2.5 Monte-Carlo Tree Search for Go

Moderate projects (60p)

Goal

Simulate computer Go game using move prediction performed using Monte-Carlo Tree search method.

Details

Input Go board

Output Proposed move (in a loop)

The general idea behind this algorithm is to choose next move without any position valuating functions. There is no strategy necessary for this method. From a given board we generate random plays and then calculate wins/looses ratio in subtrees. This ratio is a basis for next move selection.
Simulate a game computer vs computer and allow game with a human.
Due to complexity of Go game, size of the board may be decreased, for example 9x9.

2.6 Hamming one

Moderate projects (60p)

Goal

Having a set of long bit sequences find all pairs with the Hamming distance equal to 1.

Details

Input Generated special set of vectors. Sequences should be at least of thousands bits long ($l \geqslant 1000$). There should be at least 100k vectors ($n \geqslant 10^5$). The same set of inputs should be used in all tests.

Output List of pairs, tests against precomputed and expected results.

The basic algorithm performs one-to-all checking in $O(n^2 l)$ time but it is trivial. Your solution should implement $O(nl^2)$ method based on a tree. Proper data encoding, parallel threads and warps should be used in order to improve data reading scheme and utilize cache and coalesced read. Internal bit functions should be used to perform word to word comparison.

Extra

Implementation of an algorithm with better time complexity $O(nl)$ will give 40p extra.

2.7 Levenshtein distance

Moderate projects (60p)

Goal

Implement parallel method for *lev* function. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0186251

Details

Input Two strings s_1 and s_2.

Output List of transformations to be done in order to transform s_1 to s_2.

Materiały sponsorowane przez:

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Fundusze Europejskie
Wiedza Edukacja Rozwój

Rzeczpospolita Polska

Politechnika Warszawska

Unia Europejska
Europejski Fundusz Społeczny