Obliczenia superkomputerowe - HPC (Info II) 18/19

NAWIGACJA

Kokpit
- Strona główna
  - Strony
  - Moje kursy
    - TA.ZOB.19/20
    - HPC.INFOII.18/19
      - Uczestnicy
      - Odznaki
      - Kompetencje
      - Oceny
      - Sekcja ogólna
      - Introduction to HPC
      - OpenMP
      - CUDA
        - lecture notes: 02 - CUDA
        - Lab 3 - CUDA basics
        - Lab 4 - CUDA memory types, synchronisation
        - Lab 5 - advanced features of CUDA

CUDA assignment 2019

Massively Parallel A* Search on a GPU

1. Introduction

A* (or AStar) is a graph search algorithm widely used in artificial intelligence. While the traditional version of this algorithm is purely sequential, there have been several parallel versions over the years. One such variant is the approach described by *Zhou* and *Zeng* in their paper *Massively Parallel A* Search on a GPU*. It's a version of A* designed for GPU execution. Your task in this assignment is to implement that algorightm with CUDA.

2. Details

The algorithm is described in this paper: *Massively Parallel A* Search on a GPU*.

The paper describes the search algorithm itself and two node duplication detection schemes: *Parallel Cuckoo Hashing* and *Parallel Hashing with Replacement.* Duplication detection is needed so the A* algorithm doesn't expand already visited nodes. In your solutions you need to focus only on *Parallel Hashing with Replacement* - it's a bit easier to implement. You can find the pseudocode for that duplication detection scheme in the appendix (or below in Figure 2).

Algorithm 1 GA*: Parallel A* search on a GPU

procedure GA*(s, t, k) ▷ find the shortest path from s to t with k queues
    Let $\{Q_i\}_{i=1}^{k}$ be the priority queues of the open list
    Let H be the closed list
    PUSH(Q_1, s)
    $m \leftarrow$ nil ▷ m stores the best target state
    while Q is not empty **do**
        Let S be an empty list
        for $i \leftarrow 1$ to k <u>in parallel</u> **do**
            if Q_i is empty **then**
                continue
            end if
            $q_i \leftarrow$ EXTRACT(Q_i)
            if q_i.node $= t$ **then**
                if $m =$ nil **or** $f(q_i) < f(m)$ **then**
                    $m \leftarrow q_i$
                end if
                continue
            end if
            $S \leftarrow S +$ EXPAND(q_i)
        end for
        if $m \neq$ nil **and** $f(m) \leq \min_{q \in Q} f(q)$ **then**
            return the path generated from m
        end if
        $T \leftarrow S$
        for $s' \in S$ <u>in parallel</u> **do**
            if s'.node $\in H$ **and** $H[s'$.node$].g < s'.g$ **then**
                remove s' from T
            end if
        end for
        for $t' \in T$ <u>in parallel</u> **do**
            $t'.f \leftarrow f(t')$
            Push t' to one of priority queues
            $H[t'$.node$] \leftarrow t'$
        end for
    end while
end procedure

Lab activity scores
MRJP.INFOII.18/19
PSZI.INFO.III.17/18
JiPP.INFO.III.17/18
ZPP.INFO.III.17/18
IO.INFO.II.16/17
JNPI.INFO.II.16/17
BD.INFO.II.16/17

ADMINISTRACJA

Administracja kursem

Algorithm A3 Parallel hashing with replacement on a GPU

```
procedure HASH-WITH-REPLACEMENT-DEDUPLICATE(H, T)
    T' ← T
    for i ← 0 to |T| in parallel do
        z ← 0
        for j ← 0 to d − 1 do
            if H[h_j(T[i])] ∈ {T[i], nil} then
                z ← j
                break
            end if
        end for
        t ← T[i]
        ATOMIC-SWAP(t, H[h_z(T[i])])
        if t = T[i] then
            remove T[i] from T'
            continue
        end if
        for j ← 0 to d − 1 do
            if j ≠ z and H[h_j(T[i])] = T[i] then
                remove T[i] from T'
                break
            end if
        end for
    end for
    return T'
end procedure
```

Figure 1. Pseudocode for the parallel A* algorithm **Figure 2**. Pseudocode for Parallel Hashing with Replacement

The paper presents 3 possible applications of A*: sliding puzzles, pathfinding and protein design. In this assignment we'll focus on the first two.

a) Sliding puzzles

A sliding puzzle is a task of finding the smallest number of moves from one configuration of cells to another in an NxN grid, where each move slides one cell to an empty space in the grid. Here we'll use $N = 5$.

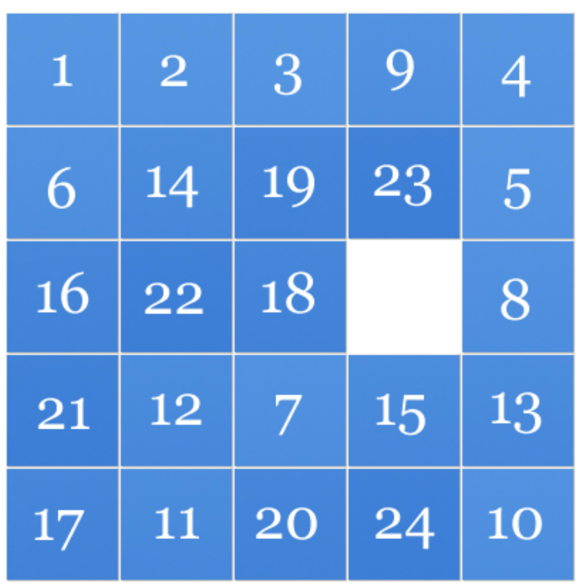

Figure 3. A $5x5$ sliding puzzle

We need to use a heuristic function that approximates a distance to the target configuration. The *Zhou* paper uses the *disjoint pattern database* method, but we'll use a simple *Manhattan* distance instead. Let's look at an example for a $3x3$ grid:

```
2 1 3     1 2 3         1 2 3 4 5 6 7 8
5 4 0     4 5 6         ---------------
6 7 8     7 8 0         1+1+0+1+1+3+1+1 = 9
```

For each cell we count the number of moves necessary to get to their target positions (or simply $|x_1 - x_2| + |y_1 - y_2|$, where x_i, y_i are cell coordinates) and sum them together. The resulting number is an approximate distance between two configurations.

b) Pathfinding

In this task we search for a shortest path between two points on a grid. Each cell is connected with up to 8 neighbours and connections are weighted. One example of a task like this is moving a unit in the *Civilization* video game. The map in that game is comprised of square tiles. Each of them has a specific terrain type which has an effect on movement speed. For instance, moving through mountains is more difficult than over plains, so the movement cost is higher in the former case.

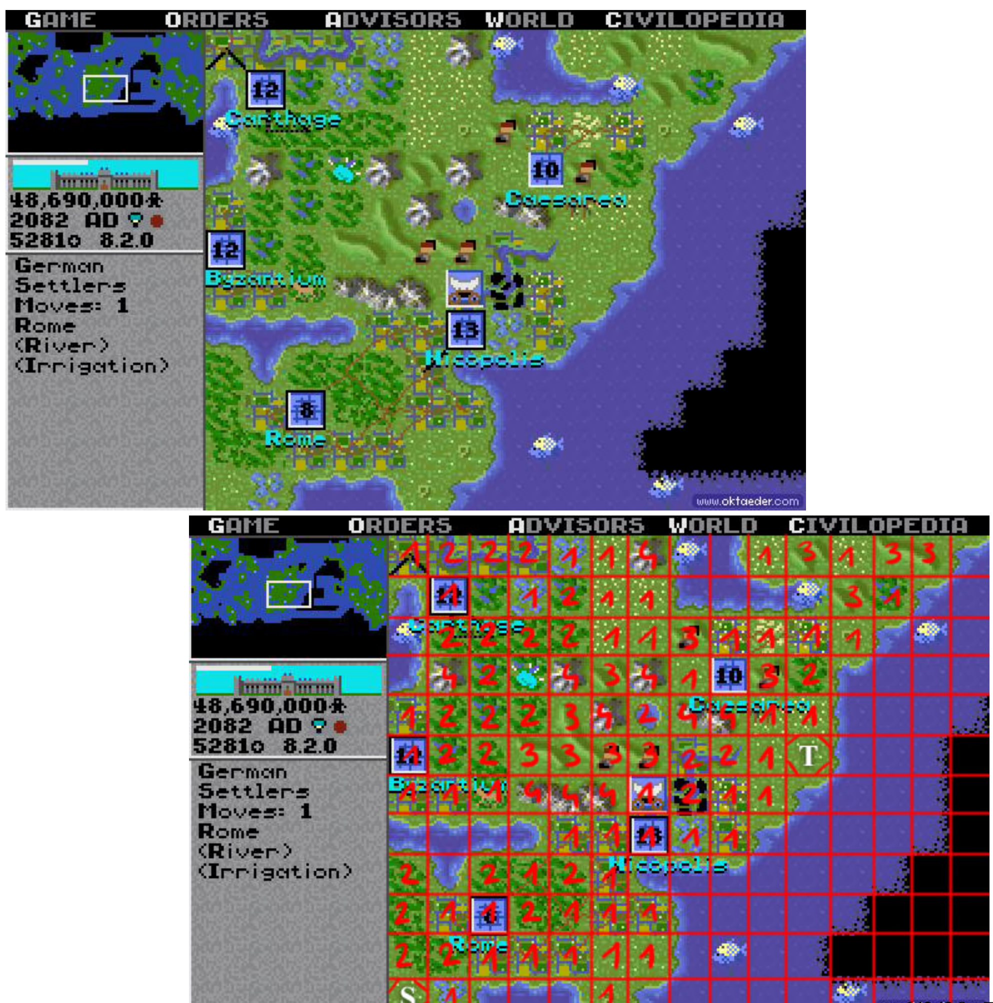

Figure 4. Map view in the original *Civilization*

Figure 5. Map view with an overlay indicating: start and target positions. movement costs

In this assignment, all weights are 1 unless they're overridden in the input file. Similarly, it's possible to set obstacles in the input file - nodes don't form connections to cells denoted as such. Cell at position $(0, 0)$ is in the lower left corner of the grid. Weight values are integers, ranging from 1 to 5. The maximum grid size is 10000 x 10000. Cell coordinates are formatted as (x, y).

For the heuristic function, we again use the *Manhattan* distance: $|x_1 - x_2| + |y_1 - y_2|$.

3. Requirements

Write your kernel code in CUDA C. The host code can be either in C or C++. The specifics of the device code are up to you - choose appropriate memory types, synchronisation techniques, use streaming or not, etc. You can optimise for Titan X GPUs on *bruce*. Use one GPU - you don't have to utilise a multi-GPU setup.

IMPORTANT: You can't use third-party libraries in your kernels - all code that's executed on the GPU needs to be implemented by you. You can however use external libraries in your host code to handle I/O tasks, prepare input data, etc.

You can organise your source code in any way you like, but it has to be properly divided into modules. Your Makefile should produce a single executable named astar_gpu.

That executable must handle the following command line arguments:
--version sliding | pathfinding - selects either sliding puzzle or pathfinding
--input-data *PATH* - file containing graph definition
--output-data PATH - file containing program execution time in milliseconds (excluding I/O operations) and search results

You should describe your solution in a detailed report - focus on explaining your implementation and optimisation techniques. Include screenshots from NVIDIA Visual Profiler that show GPU utilisation during program execution.

Send your solution as a zip file named ab123456_CUDA.zip, where ab123456 is your login (use this form). It should have the following contents:

```
ab123456_CUDA/
    source/
        src_file1.cu
        src_file2.cu
        ...
        Makefile
    report/
        ab123456_report.pdf
```

a) Sliding puzzle

./astar_gpu --version sliding --input-data /home/user/input.txt --output-data /home/user/result.txt

For the sliding puzzle, the input file must have the following format:

```
1,2,_,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
24
4,9,8,12,14,_,18,13,11,19,21,23,20,24,15,1,3,7,16,5,6,2,10,17,
22
```

The first line is the start configuration and the second line is the target configuration.

The output file should look like this:

```
1324
1,2,_,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
24
1,2,3,_,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
24
...
4,9,8,12,14,18,_,13,11,19,21,23,20,24,15,1,3,7,16,5,6,2,10,17,
22
4,9,8,12,14,_,18,13,11,19,21,23,20,24,15,1,3,7,16,5,6,2,10,17,
22
```

The first line is the execution time in milliseconds. The following lines are the shortest path found by the A* algorithm.

b) Pathfinding

./astar_gpu --version pathfinding --input-data /home/user/input.txt --output-data /home/user/result.txt

For pathfinding, the input file must have the following format:

```
10,10
0,0
9,9
2
1,1
2,3
4
2,2,3
4,3,2
1,4,5
3,4,2
```

The first line is the grid size. The second line is the start position, followed by the end position in the third line. The fourth line is the number of obstacle positions, o. Then there are o lines with obstacle coordinates. This is followed by a line containing the number of cells which have connections with weights different than 1, w. Then there are w lines, each storing the cell coordinates and a weight value.

The output file should look like this:

```
2345
0,0
0,1
...
8,9
9,9
```

The first line is the execution time in milliseconds. The following lines are the shortest path found by the A* algorithm.

c) Path doesn't exist

If the algorithm can't find a solution (either in the sliding puzzle or pathfinding task), the program should finish properly and create an output file that contains only the execution time in milliseconds.

4, Example data

a) Sliding puzzle

Input: puzzle_input.txt

Output: puzzle_output.txt

b) Pathfinding

Input: pathfinding_input.txt

Output: pathfinding_output.txt

5. Frequently Asked Questions

1) What's the submission deadline?

28.04.2019 23:55

2) What is the upper limit for the grid size in the pathfinding problem?

10000 x 10000 cells, for more details see section **2b**.

Edit history:

29.03.2019: Initial version

15.04.2019: Added limits for graph weights and grid size in the pathfinding problem. Added FAQ.

19.04.2019: Added example input and output data. Added requirements for the case when the path doesn't exist.

Ostatnia modyfikacja: piątek, 19 kwiecień 2019, 21:09

◄ Lab 6 - CUDA debugging, profiling and optimisation

Przejdź do...

CUDA assignement - solution form ►