Michał Piotr Stankiewicz
PWiR
MIMUW 2016/2017
335789
Task 2

Raport

I have solved a given taks to me using C++ and MPI.
I have implemented 2 given algorithms, and one mine.

My algorithm:

That algorithm is simple. We dived data over processes in equal number. To do so, i-th process get stars i + k * PROCESSES_NUMBER. In that case split is „equal”, but we need to provide all processes with all initial data.

Bottleneck of algorithms:

All algorithms has its own bottle neck which is redistributing data. The longest one is first, initial distribution.
Algorithm 1) and 3) needs all data, so that takes a long time.
Algorithm 2) need only 9 pieces of data. Potentially it can be noting, all or 9 * DATA_SIZE / PROCESSES_NUMBER.

Than comes redistributing it back.
Algorithm 1) redistributs its own chunk of data. It can be as big as whole given data, if all stars are in one cell.
Algorithm 2) redistributs its own chunk of data by, first sending information to each process how much after its own iteration there are stars to send to them, and than send these stars. Potentially it should communicate first with all processes, but it sens only list of ints of length PROCESSES_NUMBER, than it should send and receive data between up to 8 other processes.
Algorithm 3) redistributs its own data by sending its own chunk to all other processes and receiving their chunks. Size of chunk is fixed, so less connection is needed.

Calculation
Algorithm 1)
Each process can do from 0 to n stars, but for each star it need to do calculation for n – 1 other stars.
Algorithm 2)
It calculate data between its chunk and its neighbours. For many processes chunks should be small, data can corrupted, because important stars nearby will be forgotten.
Algorithm 3)
It has always all data to its own calculation, but calculate only 1 / PROCESSES_COUNT data.

Here are some results, first on my machine. 4 Cores.

PROVIDED TESTS (alg, test, processes)

(1, one-star, 16) - 1,05s
(2, one-star, 16) - 0,66s
(3, one-star, 16) - 0,47s

(1, fifty-stars, 16) - 0,98s
(2, fifty-stars, 16) - 0,68s
(3, fifty-stars, 16) – 0,61s

That case is not very interesting mostly because there are lots of processes, but data is really small. Redestribution takes a lot of time, normally most. You can see algorithm 3 is best, but only some better than algorithm 2

(alg, processes, move, stars) – 10 ITERS
(1, 6, 1000, 2000) - 3.63s
(2, 6, 1000, 2000) - 3.48s
(3, 6, 1000, 2000) - 1.10s

(1, 36, 1000, 2000) - 10,75s
(2, 36, 1000, 2000) - 6,75s
(3, 36, 1000, 2000) - 4.45s

(1, 64, 1000, 2000) - 15,75s
(2, 36, 1000, 2000) - 10.68s
(3, 36, 1000, 2000) - 6.90s

(1, 6, 1, 2000) - 3.62s
(2, 6, 1, 2000) - 3.71s
(3, 6, 1, 2000) - 1.15s

(1, 36, 1, 2000) - 8.18s
(2, 36, 1, 2000) - 6.32s
(3, 36, 1, 2000) - 4.48s

(1, 64, 1, 2000) - 12.58s
(2, 64, 1, 2000) - 9.86s
(3, 64, 1, 2000) - 7.25s

(1, 6, 1, 4050) - 13.86s
(2, 6, 1, 4050) - 12.96s
(3, 6, 1, 4050) - 5.04s

(1, 36, 1, 4050) - 33.50s
(2, 36, 1, 4050) - 18.02s
(3, 36, 1, 4050) - 17.66s

(1, 64, 1, 4050) - 40.06s
(2, 64, 1, 4050) - 31.20s
(3, 64, 1, 4050) – 22.49s

Here also algorithm 3) win, and is sometimes way batter than algorithm 2), always better than 1).

MIMUW LAB - 10 iters

(1, 64, 100, 11250) - 94.38s
(2, 64, 100, 11250) - 50.54s
(3, 64, 100, 11250) - 37.42s

Here, for bigger dataset, nearly 3 times bigger, algorithm 3) is about 80% slower only, algorithm 2 only about 60% and algorithm 1) more than 130%. As far as data is about 3 times bigger, if we were doing it on one processor, time should be about 9 times bigger.

Below is bigger example. - 250 iterations, so redistributing initial data is less important

(1, 81, 1, 16422) – 1522.05s
(2, 81, 1, 16422) – 267.84s
(3, 81, 1, 16422) – 459.10s

Here is an example where 3) looses with 2). For data very close to itself.

Algorith 2) should be useless the way it is if the intial galaxies are far away with them.