ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

CS342 - Parallel Programming

Spring Semester 2023-2024

3rd & 4th Assignments Report

Barnes-Hut Simulation using TBB & Java Threads

Κουμάκης Εμμανουήλ
AM : 4281

1. Compilation - Execution Instructions

How to compile:

- CPP program : make _cpp
- Java program: make _java

- You can also use “make all” to compile both versions at the same time.

How to run:

- CPP program: ./BarnesHut <input_file> #iterations #threads <output_file>
- Java program: java Main <input_file> #iterations #threads <output_file>

2. Implementation

2.1 Barnes-Hut Algorithm:

- Αρχικά διαβάζουμε τα σώματα από το αρχείο και τα αποθηκεύουμε μαζί με όλα τους τα στοιχεία σε ένα δυναμικό πίνακα (arraylist ή vector αντίστοιχα).
- **Barnes-Hut Tree Construction :** Για τη δημιουργία του δέντρου παίρνουμε διαδοχικά τα αποθηκευμένα σώματα και για κάθε ένα σώμα **b** ακολουθούμε την παρακάτω διαδοχική διαδικασία.
    - Αν ο κόμβος που βρισκόμαστε δεν περιέχει κάποιο σώμα, τότε εισάγουμε το **b** σε αυτόν και επιστρέφουμε.
    - Αν ο κόμβος περιέχει ήδη κάποιο σώμα τότε :
        - Αν ο κόμβος είναι φύλλο, πάει να πει ότι παραπάνω από ένα σώματα αντιστοιχούν στο συγκεκριμένο quad και συνεπώς πρέπει να διαιρέσουμε. Έτσι δημιουργούμε 4εις υπό-κόμβους και στην συνέχεια βρίσκουμε και εισάγουμε αναδρομικά το **b**, καθώς και το σώμα που υπήρχε ήδη στον (πλέον εσωτερικό) κόμβο, στο κατάλληλο φύλλο-κόμβο (quad) που αντιστοιχούν.

        - Αν είναι εσωτερικός κόμβος (όχι φύλλο), υπολογίζουμε και ανανεώνουμε το κέντρο μάζας του σώματος που υπάρχει ήδη στον κόμβο (dummy πλέον σώμα που κρατάει το κέντρο μάζας του quadrant που αντιστοιχεί ο κόμβος). Στην συνέχεια

βρίσκουμε αναδρομικά τον κατάλληλο κόμβο (quad) που πρέπει να μπει το **b**.

- **Net-Force Calculation:** Όσον αφορά τον υπολογισμό της συνολικής δύναμης που ασκείται σε κάθε σώμα, ακολούθησα λίγο διαφορετική προσέγγιση, ως προς το πως καθορίζεται αν ο εσωτερικός κόμβος (το κέντρο μάζας του) βρίσκεται σε αρκετά μεγάλη απόσταση από το σώμα **b**, σε σχέση με αυτό που περιγράφεται στην εκφώνηση της άσκησης, καθώς αυτό που αναφέρεται στην εκφώνηση με μπέρδευε και δυσκολευόμουν να το καταλάβω πλήρως.
  Για να προσδιορίσουμε αν το κέντρο μάζας απέχει αρκετά από το σώμα **b** υπολογίζουμε το πηλίκο s/d, όπου s είναι το μέγεθος του quad (εσωτερικού κόμβου) και d είναι η απόσταση μεταξύ του κέντρου βάρους του εσωτερικού κόμβου και του σώματος **b**. Αν αυτό το πηλίκο είναι μικρότερο από ένα threshold, πάει να πει ότι το σώμα **b** βρίσκεται σε αρκετά μεγάλη απόσταση από τον εσωτερικό κόμβο και μπορούμε να υπολογίσουμε τη συνολική δύναμη που ασκείται στο **b** από όλα τα σώματα που ανήκουν στον εσωτερικό κόμβο μέσω του 'snapshot' του κέντρου μάζας. Όσο πιο μικρό είναι threshold που αναφέρθηκε, τόσο αυξάνεται η ακρίβεια της προσομοίωσης. Για τη δική μας προσομοίωση χρησιμοποιήθηκε threshold=0.5.
  Επομένως για τον υπολογισμό της δύναμης που ασκείται σε κάθε σώμα **b** ξεκινάμε από τη ρίζα του δέντρου ακολουθούμε την παρακάτω διαδικασία:
  - Αν ο κόμβος που βρισκόμαστε είναι φύλλο, τότε αν το σώμα που περιέχει είναι διαφορετικό από το σώμα **b** (για το οποίο υπολογίζουμε την δύναμη), τότε υπολογίζουμε και ανανεώνουμε την δύναμη που ασκείται στο **b**.
  - Αν ο κόμβος είναι εσωτερικός (όχι φύλλο), τοτε υπολογίζουμε το πηλίκο s/d που περιγράφεται παραπάνω και :
    - Αν είναι μικρότερο από το threshold που έχουμε ορίσει (0.5), τότε πάει να πει ότι ο εσωτερικός κόμβος βρίσκεται αρκετά μακριά από το σώμα **b** και επομένως υπολογίζουμε τη δύναμη που ασκείται στο **b** από όλα τα σώματα που ανήκουν στον εσωτερικό κόμβο μέσω του 'snapshot' του κέντρου μάζας αυτου του κόμβου (quad).
    - Αν δεν είναι μικρότερο από το threshold, τότε διασχίζουμε αναδρομικά κάθε κόμβο παιδί του τρέχων εσωτερικού κόμβου για συνεχιστεί η διαδικασία.

  Ο υπολογισμός της δύναμης καθώς και η νέα θέση καθορίζεται ακριβώς όπως περιγράφεται στην εκφώνηση.

Τόσο στην κατανόηση αλλά και στην υλοποίηση του αλγορίθμου για τη δημιουργία του δέντρου αλλά και τον υπολογισμό της συνολικής δύναμης που ασκείται σε κάθε σώμα, βοήθησε αρκετά αυτό το άρθρο των Tom Ventimiglia & Kevin Wayne.

2.2 Parallelism :

Όσον αφορά το παράλληλο κομμάτι του προγράμματος και στις δύο υλοποιήσεις παραλληλοποιούμε την δεύτερη και τρίτη φάση του αλγορίθμου (δηλαδή τον υπολογισμό της συνολικής δύναμης που ασκείται σε κάθε σώμα & τον υπολογισμό της νέας θέσης κάθε σώματος).

2.2.1 Παραλληλισμός με Java:

Αφού έχουμε δημιουργήσει (σειριακά) το δέντρο, “μοιράζουμε” ισόποσα τα σώματα στα threads και σε κάθε ένα εκτελούμε την δεύτερη φάση του αλγορίθμου. Στη συνέχεια χρησιμοποιούμε συγχρονισμό για να βεβαιωθούμε ότι όλα τα threads έχουν ολοκληρώσει τον υπολογισμό της δύναμης στα σώματα και τέλος να προχωρήσουμε στην εκτέλεση της τρίτης φάσης του αλγορίθμου.
Για συγχρονισμό χρησιμοποιούμε ένα Cyclicbarrier, το οποίο καλεί την μέθοδο await για κάθε thread που φτάνει στο σημείο που έχουμε ορίσει (στο τέλος της δεύτερης φάσης). Όταν και το τελευταίο thread φτάσει σε αυτό το σημείο, τότε το barrier γίνεται release και τα threads προχωράνε στον υπολογισμός της νέας θέσης.

3. Time Statistics

1. Για τις μετρήσεις του χρόνου εκτέλεσης και στις δύο υλοποιήσεις χρησιμοποιήθηκε η εντολή time του shell.
2. Κάθε input file έχει δοκιμαστεί για 10.000 και 100.000 iterations.
3. Όλα τα παρακάτω στατιστικά είναι μετρημένα σε δευτερόλεπτα.
4. Για τον υπολογισμό του μέσω χρόνου εκτέλεσης, της διακύμανσης, και του speedup χρησιμοποιήθηκε ένα python script με την βιβλιοθήκη statistics.
5. Όσον αφορά την διακύμανση των χρόνων εκτέλεσης υπολογίζουμε το standard deviation (τυπική απόκλιση, όχι variance) καθώς το standard deviation είναι στην ίδια μονάδα μέτρησης με τα δεδομένα μας.

3.1 C++ with TBB Statistics:

- **Input1 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	0,161s	0,157s	0,169s	0,155s	0,155s	0,159s	0,006	1
1	0,213s	0,204s	0,203s	0,234s	0,225s	0,216s	0,013	0,737
2	0,231s	0,226s	0,200s	0,193s	0,254s	0,221s	0,025	0,720
4	0,261s	0,236s	0,261s	0,264s	0,274s	0,259s	0,014	0,613

- **Input1 x 100000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	1,648s	1,731s	1,472s	1,538s	1,530s	1,584s	0,104	1
1	1,975s	1,565s	1,408s	1,360s	1,425s	1,547s	0,251	1,024
2	1,550s	1,519s	1,469s	1,367s	1,371s	1,455s	0,084	1,089
4	1,814s	1,802s	1,864s	2,228s	2,154s	1,972s	0,203	0,803

- **Input2 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	6,401s	6,983s	8,431s	7,067s	7,65s	7,306s	0,769	1
1	7,875s	8,792s	8,336s	8,094s	7,541s	8,128s	0,473	0,899
2	7,518s	7,844s	8,367s	7,723s	8,136s	7,918s	0,336	0,923
4	6,192s	6,095s	5,786s	6,025s	5,872s	5,994s	0,165	1,219

- **Input2 x 100000**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	81,166s	81,716s	82,356s	80,69s	82,472s	81,680s	0,763	1
1	78,009s	79,908s	76,762s	76,661s	76,071s	77,482s	1,529	1,054
2	66,781s	65,487s	65,617s	65,833s	65,614s	65,866s	0,526	1,240
4	60,143s	59,113s	59,808s	58,778s	58,37s	0,729	1,379	1,379

- **Input3 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	5,756s	7,178s	6,267s	6,835s	5,476s	6,302s	0,713	1
1	6,027s	5,694s	6,085s	7,32s	6,185s	6,262s	0,619	1,006
2	5,971s	5,281s	5,548s	5,238s	5,641s	5,536s	0,298	1,138
4	5,623s	5,102s	4,718s	5,162s	5,08s	5,137s	0,323	1,227

- **Input3 x 100000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	62,26s	64,571s	59,148s	61,031s	59,883s	61,379s	2,139	1
1	64,837s	65,74s	67,673s	66,495s	67,089s	66,367s	1,115	0,925
2	52,839s	54,046s	53,281s	53,706s	53,787s	53,532s	0,475	1,147
4	48,74s	48,583s	48,580s	48,533s	48,430s	48,573s	0,112	1,264

- **Input4 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	95,971s	85,017s	96,661s	91,0s	96,168s	92,963s	5,000	1
1	103,68s	96,298s	126,74s	106,76s	105,04s	107,709s	11,367	0,863
2	88,493s	89,583s	92,343s	91,737s	86,322s	89,696s	2,450	1,036
4	76,895s	79,723s	76,533s	78,816s	79,496s	78,293s	1,485	1,187

- **Input4 x 100000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	1201,2s	1189,4s	1210,8s	1208,2s	1182,5s	1198,43s	12,147	1
1	1250,4s	1247,2s	1255,1s	1248,8s	1245,3s	1249,3s	3,682	0,959
2	891,82s	889,48s	886,13s	893,52s	879,96s	888,188s	5,365	1,349
4	739,09s	726,88s	714,24s	722,02s	737,32s	727,915s	10,442	1,646

- **Input5 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	176,95s	167,16s	169,13s	174,43s	178,45s	173,229s	4,905	1
1	176,15s	182,15s	184,54s	177,03s	185,84s	181,144s	4,372	0,956
2	124,07s	127,14s	124,35s	122,8s	126,28s	124,932s	1,755	1,387
4	102,08s	102,68s	102,5s	101,95s	99,3s	101,704s	1,375	1,703

❖ **Input5 x 100000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	1911,8s	1927,6s	1942,9s	1918,6s	1915,1s	1923,22s	12,502	1
1	1960,9s	1922,3s	1943,1s	1947,0s	1919,8s	1938,65s	17,375	0,992
2	1419,6s	1431,2s	1435,7s	1422,1s	1425,2s	1426,80s	6,633	1,348
4	1122,5s	1112,7s	1107,3s	1125,5s	1119,8s	1117,571s	7,434	1,720

3.1 Java Statistics:

❖ **Input1 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	0,219s	0,231s	0,241s	0,244s	0,402s	0,267s	0,076	1
1	1,668s	1,681s	1,771s	1,723s	1,678s	1,704s	0,043	0,157
2	2,636s	2,,690s	2,688s	2,661s	2,668s	2,669s	0,022	0,100
4	4,811s	4,828s	4,924s	4,940s	5,855s	5,072s	0,442	0,053

❖ **Input1 x 100000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	0,406s	0,405s	0,438s	0,473s	0,415s	0,427s	0,029	1
1	14,350s	14,574s	14,489s	16,351s	14,354s	14,824s	0,859	0,029
2	29,343s	27,545s	24,465s	25,265s	26,448s	22,243s	10,841	0,019
4	51,354s	56,35s	53,7s	60,377s	55,456s	55,447s	3,351	0,008

- **Input2 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	0,794s	1,046s	0,906s	1,113s	1,035s	0,979s	0,128	1
1	2,048s	2,050s	2,066s	2,048s	2,036s	2,05s	0,011	0,478
2	3,093s	2,654s	2,582s	3,287s	2,728s	2,869s	0,306	0,341
4	4,246s	4,234s	4,331s	4,336s	4,737s	4,377s	0,207	0,224

- **Input2 x 100000**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	7,784s	7,241s	6,589s	7,196s	7,131s	7,188s	0,424	1
1	20,504s	20,184s	20,922s	20,805s	21,413s	20,766s	0,461	0,346
2	26,566s	26,235s	26,541s	25,883s	26,028s	26,251s	0,304	0,274
4	48,233s	46,758s	45,933s	45,718s	47,663s	46,861s	1,084	0,153

- **Input3 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	0,689s	0,683s	0,807s	0,850s	0,873s	0,780s	0,089	1
1	1,802s	2,227s	2,002s	1,928s	1,840s	1,960s	0,168	0,398
2	2,920s	2,736s	2,645s	2,655s	2,732s	2,738s	0,110	0,285
4	5,151s	5,007s	4,,224s	4,831s	4,965s	4,836s	0,360	0,161

- **Input3 x 100000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	5,446s	5,768s	5,973s	5,7s	5,36s	5,649s	0,248	1
1	19,221s	16,726s	18,416s	16,596s	17,352s	17,662s	1,130	0,320
2	24,551s	25,847s	24,451s	25,127s	25,107s	25,017s	0,558	0,226
4	42,636s	43,262s	43,418s	44,381s	43,631s	43,466s	0,632	0,130

❖ **Input4 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	11,869s	9,745s	11,623s	10,788s	10,827s	10,970s	0,835	1
1	12,983s	14,418s	14,920s	14,261s	13,115s	13,939s	0,850	0,787
2	10,011s	11,001s	9,052s	9,644s	11,205s	10,183s	0,910	1,077
4	10,488s	11,822s	11,019s	11,002s	11,017s	11,070s	0,478	0,991

❖ **Input4 x 100000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	108,55s	108,85s	109,91s	110,31s	107,24s	108,972s	1,211	1
1	132,67s	135,19s	134,17s	135,3s	135,46s	134,561s	1,171	0,810
2	107,13s	101,42s	99,53s	109,01s	107,3s	104,890s	4,151	1,039
4	100,9s	99,846s	106,64s	106,29s	106,98s	104,134s	3,46	1,046

❖ **Input5 x 10000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	19,881s	19,083s	17,844s	16,792s	17,671s	18,254s	1,223	1
1	25,407s	25,425s	25,324s	25,446s	25,128s	25,346s	0,130	0,720
2	17,138s	18,479s	17,852s	17,537s	17,557s	17,713s	0,498	1,031
4	17,277s	17,382s	17,064s	16,926s	17,970s	17,324s	0,403	1,054

❖ **Input5 x 100000 iterations**

#Threads	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time	Std Dev.	Avg Speedup
0 (serial)	177,32s	167,85s	167,79s	167,55s	174,51s	171,009s	4,594	1
1	192,83s	192,43s	195,93s	190,26s	191,35s	192,564s	2,132	0,888
2	167,90	166,32s	163,73s	166,86s	165,28s	166,021s	1,593	1,030
4	140,25s	142,19s	152,3s	133,04s	128,41s	139,24s	9,166	1,228

4. Implementations Comparison

Γενικές δυσκολίες που αντιμετώπισα:
1. Μέχρι να καταλάβω το πως πρέπει να χωρίζεται σωστά το κάθε κάθε quad σε υπό-quads, καθώς το μέγεθος του σύμπαντος R που διαβάζουμε από το αρχείο αντιστοιχεί στο 'μήκος' μιας πλευράς και όχι στο 2R (από -R έως R) που είναι το πραγματικό μέγεθος του σύμπαντος.
2. Η δυσκολία στην κατανόηση για το πως καθορίζεται αν ο εσωτερικός κόμβος (το κέντρο μάζας του) βρίσκεται σε αρκετά μεγάλη απόσταση από το σώμα **b,** όπως περιγράφεται και παραπάνω στην προηγούμενη ενότητα.

Σχεδιασμός :
Ο σχεδιασμός και στις δύο υλοποιήσεις όσον αφορά τις κλάσεις που χρησιμοποιούμε για να αναπαραστήσουμε τα σώματα, το δέντρο κλπ είναι ακριβώς ο ίδιος. Ο αλγόριθμος που χρησιμοποιήθηκε είναι εξίσω ο ίδιος προσαρμοσμένος στα χαρακτηριστικά της κάθε γλώσσας με τις δομές που παρέχονται. (π.χ στην Java χρησιμοποιούμε ArrayList για να αποθηκεύουμε τα σώματα ενώ στην C++ χρησιμοποιήσαμε Vector).

Ευκολία Υλοποίησης:
Ξεκίνησα την υλοποίηση της άσκησης με Java καθώς είναι μια γλώσσα που είχα αρκετή τριβή στο παρελθόν και συνεπώς δεν αντιμετώπισα κάποια δυσκολία όσον αφορά το προγραμματιστικό μοντέλο. Από την άλλη η C++ είναι μια γλώσσα που δεν είχα σχεδόν καμία επαφή και χρειαζόμουν την βοήθεια του google για να βρω κάποια πράγματα. Παρόλα αυτά, έχοντας έτοιμη την υλοποίηση σε Java, η μετατροπή δεν ήταν ιδιαίτερα δύσκολη.
Η κύρια δυσκολία που αντιμετώπισα στην υλοποίηση με C++ ήταν το πως έπρεπε να αποδεσμεύσω σωστά τη μνήμη ώστε να μην crasharei το πρόγραμμα στα μεγάλα input για 100k iterations.

Ταχύτητα Εκτέλεσης:
Όσον αφορά την διαφορά στην ταχύτητα εκτέλεσης ανάμεσα στις δύο υλοποιήσεις παρατηρούμε ότι στην υλοποίηση με Java, τόσο στα μικρά input sizes (εκτός του inpu1) όσο και στα μεγάλα, η ταχύτητα εκτέλεσης είναι **πολύ** γρηγορότερη σε σχέση με αυτή της C++. Παρόλα αυτά παρατηρούμε επίσης ότι για μικρά input sizes το speedup μειώνεται όσο αυξάνεται ο αριθμός των threads. Στα μεγάλα input sizes βλέπουμε για πρώτη φορά αύξηση στο speedup όσο αυξάνεται ο αριθμός των threads, αλλά αυτή η αύξηση δεν είναι σπουδαία. Από την άλλη στην υλοποίηση με C++ παρατηρούμε αύξηση στο speedup, όσο αυξάνεται ο αριθμός των threads, με την αύξηση να είναι όλο και πιο σπουδαία όσο μεγαλώνει ο αριθμός των iterations ακόμη και για τα μικρά input sizes.
Έτσι συμπεραίνουμε η υλοποίηση με Java είναι πολύ πιο γρήγορη σε σχέση με την υλοποίηση σε C++, ειδικότερα στα μεγάλα input sizes. Όμως το overhead που προστίθεται όσο αυξάνονται τα threads είναι αρκετά μεγάλο ώστε να επηρεάζεται αρνητικά το performance του προγράμματος μας. Από την άλλη η υλοποίηση με C++ είναι σημαντικά πιο αργή, όμως με την αύξηση στον αριθμό των threads βελτιώνεται αρκετά το performance του προγράμματος.