Εργασία 2

Ονοματεπώνυμο: Κωνσταντίνος Λάρκου
Αριθμός Ταυτότητας: 1011182

Ερώτημα:
1) Συναρτήσεις για `n_body_omp_static`. (10% Κώδικας + 5% Σχόλια) (`Courier New 10`):

```
void computeAccelerations_static(int thread_num)
{
    int i, j;

    // Parallelize the outer loop using OpenMP
    // Each thread gets a chunk of iterations based on the number of threads and the total number of bodies
#pragma omp parallel for private(i, j) shared(accelerations) schedule(static, bodies / thread_num) num_threads(thread_num)
    for (i = 0; i < bodies; i++)
    {
        // Reset the acceleration vector for the current particle
        accelerations[i].x = 0;
        accelerations[i].y = 0;
        accelerations[i].z = 0;

        // Compute the acceleration on the current particle due to all other particles
        for (j = 0; j < bodies; j++)
        {
            if (i != j)
            {
                // Compute the distance vector and its magnitude between the current particle and the other particle
                vector sij = {positions[i].x - positions[j].x, positions[i].y - positions[j].y, positions[i].z - positions[j].z};
                vector sji = {positions[j].x - positions[i].x, positions[j].y - positions[i].y, positions[j].z - positions[i].z};
                double mod = sqrt(sij.x * sij.x + sij.y * sij.y + sij.z * sij.z);

                // Compute the gravitational force between the particles and scale it by the distance vector and its magnitude
                double mod3 = mod * mod * mod;
                double s = GravConstant * masses[j] / mod3;
                vector S = {s * sji.x, s * sji.y, s * sji.z};

                // Add the scaled force to the acceleration vector of the current particle
                accelerations[i].x += S.x;
                accelerations[i].y += S.y;
                accelerations[i].z += S.z;
            }
        }
    }
}
```

```
void computeVelocities_static(int thread_num)
{
    int i;
    // Loop through each body in parallel
#pragma omp parallel for private(i) shared(velocities) schedule(static, bodies / thread_num) num_threads(thread_num)
    for (i = 0; i < bodies; i++)
    {
        // velocities[i] = addVectors(velocities[i],accelerations[i]);
        // Compute the new velocity for the current body
        vector ac = {velocities[i].x + accelerations[i].x, velocities[i].y + accelerations[i].y, velocities[i].z + accelerations[i].z};
        velocities[i] = ac;
    }
}
```

```
void computePositions_static(int thread_num)
{
    int i;
    // Define a parallel region with private variable i and shared variable positions.
    // The schedule is static with a chunk size of bodies / thread_num to evenly distribute work among threads.
    // The number of threads used is specified by thread_num.
#pragma omp parallel for private(i) shared(positions) schedule(static, bodies / thread_num) num_threads(thread_num)
    for (i = 0; i < bodies; i++)
    {
        // Calculate the new position of the ith body by adding the velocity and half of the acceleration to the current position.
        vector sc = {0.5 * accelerations[i].x, 0.5 * accelerations[i].y, 0.5 * accelerations[i].z};
        vector ac = {velocities[i].x + sc.x, velocities[i].y + sc.y, velocities[i].z + sc.z};
        vector bc = {positions[i].x + ac.x, positions[i].y + ac.y, positions[i].z + ac.z};
        positions[i] = bc;
    }
}
```

```
void simulate_static(int threads)
{
    // I dont run with the implemented computePositions and computeVelocities because the overheads outweight the parallel benefits
    SimulationTime++;
    computeAccelerations_static(threads);
    computePositions();  // computePositions_static(int threads)
    computeVelocities(); // computeVelocities_static(int threads)
    resolveCollisions_static(threads);
}
```

Για την παραλληλοποίηση της συνάρτησης computeAccelerations_static, ο εξωτερικός βρόχος for παραλληλίζεται χρησιμοποιώντας parallel for directive του OpenMP. Αυτή η οδηγία κατανέμει τις επαναλήψεις του βρόχου for μεταξύ των διαθέσιμων νημάτων, έτσι ώστε κάθε νήμα να εκτελεί ένα υποσύνολο του συνολικού αριθμού επαναλήψεων. Η private clause διασφαλίζει ότι κάθε νήμα έχει το δικό του ιδιωτικό αντίγραφο των μεταβλητών ευρετηρίου βρόχου i και j, έτσι ώστε να μην παρεμβαίνουν μεταξύ τους στους υπολογισμούς. Η shared clause προσδιορίζει ότι ο πίνακας επιταχύνσεων πρέπει να μοιράζεται μεταξύ όλων των νημάτων. Τέλος, schedule clause with a static argument χωρίζει τις επαναλήψεις σε κομμάτια ίσου μεγέθους, το καθένα εκχωρημένο σε ένα μόνο νήμα.

Η schedule clause with static argument στο OpenMP καθορίζει μια πολιτική προγραμματισμού στατικού βρόχου όπου οι επαναλήψεις χωρίζονται σε κομμάτια ίσου μεγέθους κατά το χρόνο μεταγλώττισης και κάθε κομμάτι εκχωρείται σε ένα μόνο νήμα. Στη συνάρτηση computeAccelerations_static, η clause χρονοδιαγράμματος (static, body/thread_num) διαιρεί τις επαναλήψεις του εξωτερικού βρόχου for σε κομμάτια ίσου μεγέθους, όπου το μέγεθος του κομματιού καθορίζεται από τον αριθμό των διαθέσιμων νημάτων (thread_num). Το μέγεθος του κομματιού υπολογίζεται διαιρώντας τον συνολικό αριθμό επαναλήψεων (σώματα) με τον αριθμό των νημάτων. Η πολιτική στατικού προγραμματισμού είναι χρήσιμη όταν ο φόρτος εργασίας είναι ομοιόμορφα ισορροπημένος μεταξύ των επαναλήψεων και μπορεί να οδηγήσει σε καλή εξισορρόπηση φορτίου μεταξύ των νημάτων.

2) Συναρτήσεις για `n_body_omp_dynamic`. (10% Κώδικας + 5% Σχόλια) (`Courier New 10`):

```
void computeAccelerations_dynamic(int thread_num)
{
    int i, j;

    // Parallelize the outer loop using OpenMP
    // Each thread gets a chunk of iterations based on the number of threads and the total number of bodies
#pragma omp parallel for private(i, j) shared(accelerations) schedule(dynamic, bodies / thread_num) num_threads(thread_num)
    for (i = 0; i < bodies; i++)
    {
        // Reset the acceleration vector for the current particle
        accelerations[i].x = 0;
        accelerations[i].y = 0;
        accelerations[i].z = 0;

        // Compute the acceleration on the current particle due to all other particles
        for (j = 0; j < bodies; j++)
        {
            if (i != j)
            {
                // Compute the distance vector and its magnitude between the current particle and the other particle
                vector sij = {positions[i].x - positions[j].x, positions[i].y - positions[j].y, positions[i].z - positions[j].z};
                vector sji = {positions[j].x - positions[i].x, positions[j].y - positions[i].y, positions[j].z - positions[i].z};
                double mod = sqrt(sij.x * sij.x + sij.y * sij.y + sij.z * sij.z);

                // Compute the gravitational force between the particles and scale it by the distance vector and its magnitude
                double mod3 = mod * mod * mod;
                double s = GravConstant * masses[j] / mod3;
                vector S = {s * sji.x, s * sji.y, s * sji.z};

                // Add the scaled force to the acceleration vector of the current particle
                accelerations[i].x += S.x;
                accelerations[i].y += S.y;
                accelerations[i].z += S.z;
            }
        }
    }
}

```

```
void computeVelocities_dynamic(int thread_num)
{
    int i;
    // Loop through each body in parallel
#pragma omp parallel for private(i) shared(velocities) schedule(dynamic, bodies / thread_num) num_threads(thread_num)
    for (i = 0; i < bodies; i++)
    {
        // velocities[i] = addVectors(velocities[i],accelerations[i]);
        // Compute the new velocity for the current body
        vector ac = {velocities[i].x + accelerations[i].x, velocities[i].y + accelerations[i].y, velocities[i].z + accelerations[i].z};
        velocities[i] = ac;
    }
}
```

```
void computePositions_dynamic(int thread_num)
{
    int i;
    // Define a parallel region with private variable i and shared variable positions.
    // The schedule is static with a chunk size of bodies / thread_num to evenly distribute work among threads.
    // The number of threads used is specified by thread_num.
#pragma omp parallel for private(i) shared(positions) schedule(dynamic, bodies / thread_num) num_threads(thread_num)
    for (i = 0; i < bodies; i++)
    {
        // Calculate the new position of the ith body by adding the velocity and half of the acceleration to the current position.
        vector sc = {0.5 * accelerations[i].x, 0.5 * accelerations[i].y, 0.5 * accelerations[i].z};
        vector ac = {velocities[i].x + sc.x, velocities[i].y + sc.y, velocities[i].z + sc.z};
        vector bc = {positions[i].x + ac.x, positions[i].y + ac.y, positions[i].z + ac.z};
        positions[i] = bc;
    }
}
```

```
476 void simulate_dynamic(int threads)
477 {
478     // I dont run with the implemented computePositions and computeVelocities because the overheads outweight the parallel benefits
479     SimulationTime++;
480     computeAccelerations_dynamic(threads);
481     computePositions();  // computePositions_dynamic(int threads)
482     computeVelocities(); // computeVelocities_dynamic(int threads)
483     resolveCollisions_dynamic(threads);
484 }
```

Σε αυτόν τον κώδικα, παραλληλίζουμε τον εξωτερικό βρόχο for χρησιμοποιώντας τη δυναμική πολιτική προγραμματισμού του OpenMP. Η οδηγία pragma διευκρινίζει ότι οι επαναλήψεις βρόχου χωρίζονται δυναμικά σε κομμάτια και κάθε κομμάτι εκχωρείται σε ένα νήμα όταν είναι διαθέσιμο. Ο αριθμός των επαναλήψεων ανά κομμάτι καθορίζεται από τον runtime scheduler, ο οποίος προσαρμόζει δυναμικά το μέγεθος του κομματιού με βάση τον τρέχοντα φόρτο εργασίας κάθε νήματος.

Η clause "schedule(dynamic, body / thread_num)" προσδιορίζει ότι οι επαναλήψεις βρόχου πρέπει να προγραμματίζονται δυναμικά και κάθε νήμα πρέπει να επεξεργάζεται ένα κομμάτι εργασίας που αποτελείται από επαναλήψεις "bodies / thread_num". Αυτό σημαίνει ότι ο φόρτος εργασίας κατανέμεται δυναμικά μεταξύ των διαθέσιμων νημάτων, με κάθε νήμα να λαμβάνει ένα νέο κομμάτι εργασίας μόλις ολοκληρώσει την επεξεργασία του τρέχοντος κομματιού του.

Ο όρος "private(i, j)" ορίζει ότι κάθε νήμα πρέπει να έχει το δικό του ιδιωτικό αντίγραφο των μετρητών βρόχου i και j. Αυτό είναι απαραίτητο για την αποτροπή συνθηκών κούρσας που μπορεί να προκύψουν όταν πολλά νήματα έχουν πρόσβαση στους ίδιους μετρητές βρόχων ταυτόχρονα.

Η cluase "shared(accelerations)" προσδιορίζει ότι ο πίνακας επιταχύνσεων πρέπει να είναι κοινόχρηστος μεταξύ όλων των νημάτων. Αυτό σημαίνει ότι όλα τα νήματα έχουν πρόσβαση ανάγνωσης και εγγραφής στον πίνακα και οποιεσδήποτε αλλαγές γίνονται από ένα νήμα είναι άμεσα ορατές σε όλα τα άλλα νήματα.

Συνολικά, η πολιτική δυναμικού προγραμματισμού είναι χρήσιμη όταν ο φόρτος εργασίας κάθε επανάληψης είναι πολύ μεταβλητός, καθώς επιτρέπει στα νήματα να λειτουργούν σε κομμάτια επαναλήψεων που χρειάζονται περίπου τον ίδιο χρόνο για να εκτελεστούν. Ωστόσο, επιβαρύνεται με κάποια επιβάρυνση λόγω της ανάγκης δυναμικής προσαρμογής του μεγέθους του κομματιού, επομένως μπορεί να μην είναι πάντα η πιο αποτελεσματική επιλογή για την παραλληλοποίηση βρόχων.

3) Συναρτήσεις για `n_body_omp_guided`. (10% Κώδικας + 5% Σχόλια) (`Courier New 10`):

```
void computeAccelerations_guided(int thread_num)
{
    int i, j;

    // Parallelize the outer loop using OpenMP
    // Each thread gets a chunk of iterations based on the number of threads and the total number of bodies
#pragma omp parallel for private(i, j) shared(accelerations) schedule(guided, bodies / thread_num) num_threads(thread_num)
    for (i = 0; i < bodies; i++)
    {
        // Reset the acceleration vector for the current particle
        accelerations[i].x = 0;
        accelerations[i].y = 0;
        accelerations[i].z = 0;

        // Compute the acceleration on the current particle due to all other particles
        for (j = 0; j < bodies; j++)
        {
            if (i != j)
            {
                // Compute the distance vector and its magnitude between the current particle and the other particle
                vector sij = {positions[i].x - positions[j].x, positions[i].y - positions[j].y, positions[i].z - positions[j].z};
                vector sji = {positions[j].x - positions[i].x, positions[j].y - positions[i].y, positions[j].z - positions[i].z};
                double mod = sqrt(sij.x * sij.x + sij.y * sij.y + sij.z * sij.z);

                // Compute the gravitational force between the particles and scale it by the distance vector and its magnitude
                double mod3 = mod * mod * mod;
                double s = GravConstant * masses[j] / mod3;
                vector S = {s * sji.x, s * sji.y, s * sji.z};

                // Add the scaled force to the acceleration vector of the current particle
                accelerations[i].x += S.x;
                accelerations[i].y += S.y;
                accelerations[i].z += S.z;
            }
        }
    }
}
```

```
void computeVelocities_guided(int thread_num)
{
    int i;
    // Loop through each body in parallel
#pragma omp parallel for private(i) shared(velocities) schedule(guided, bodies / thread_num) num_threads(thread_num)
    for (i = 0; i < bodies; i++)
    {
        // velocities[i] = addVectors(velocities[i],accelerations[i]);
        // Compute the new velocity for the current body
        vector ac = {velocities[i].x + accelerations[i].x, velocities[i].y + accelerations[i].y, velocities[i].z + accelerations[i].z};
        velocities[i] = ac;
    }
}
```

```
void computePositions_guided(int thread_num)
{
    int i;
    // Define a parallel region with private variable i and shared variable positions.
    // The schedule is static with a chunk size of bodies / thread_num to evenly distribute work among threads.
    // The number of threads used is specified by thread_num.
#pragma omp parallel for private(i) shared(positions) schedule(guided, bodies / thread_num) num_threads(thread_num)
    for (i = 0; i < bodies; i++)
    {
        // Calculate the new position of the ith body by adding the velocity and half of the acceleration to the current position.
        vector sc = {0.5 * accelerations[i].x, 0.5 * accelerations[i].y, 0.5 * accelerations[i].z};
        vector ac = {velocities[i].x + sc.x, velocities[i].y + sc.y, velocities[i].z + sc.z};
        vector bc = {positions[i].x + ac.x, positions[i].y + ac.y, positions[i].z + ac.z};
        positions[i] = bc;
    }
}
```

```
486 void simulate_guided(int threads)
487 {
488     // I dont run with the implemented computePositions and computeVelocities because the overheads outweight the parallel benefits
489     SimulationTime++;
490     computeAccelerations_guided(threads);
491     computePositions();  // computePositions_guided(int threads)
492     computeVelocities(); // computeVelocities_guided(int threads)
493     resolveCollisions_guided(threads);
494 }
```

Σε αυτόν τον κώδικα, παραλληλίζουμε τη συνάρτηση computeAccelerations_guided() χρησιμοποιώντας το OpenMP με μια καθοδηγούμενη στρατηγική προγραμματισμού.

Το #pragma omp parallel for είναι μια οδηγία μεταγλωττιστή που υποδεικνύει ότι ο βρόχος που ακολουθεί θα πρέπει να εκτελείται παράλληλα. Ο όρος private(i, j) προσδιορίζει ότι κάθε νήμα πρέπει να έχει τα δικά του ιδιωτικά αντίγραφα των μετρητών βρόχου i και j. Ο όρος shared(accelerations) υποδεικνύει ότι οι επιταχύνσεις του πίνακα πρέπει να μοιράζονται μεταξύ όλων των νημάτων. Ο όρος num_threads(thread_num) καθορίζει τον αριθμό των νημάτων που θα χρησιμοποιηθούν για τον παράλληλο βρόχο.

Η clause schedule(guided, bodies / thread_num) προσδιορίζει ότι ο βρόχος πρέπει να χωριστεί σε κομμάτια μεγέθους που μειώνονται, με το αρχικό μέγεθος κομματιού να καθορίζεται από την τιμή των σωμάτων / νήμα_αριθμός. Κάθε νήμα εκτελεί ένα κομμάτι επαναλήψεων μέχρι να ολοκληρωθεί το κομμάτι του και, στη συνέχεια, ζητά το επόμενο κομμάτι από τη δεξαμενή των επαναλήψεων που απομένουν. Το μέγεθος κάθε επόμενου κομματιού καθορίζεται με βάση την ποσότητα της εργασίας που απομένει να γίνει και τον αριθμό των νημάτων που είναι ακόμα διαθέσιμα.

Συνολικά, η στρατηγική καθοδηγούμενου προγραμματισμού είναι χρήσιμη όταν η ποσότητα εργασίας που απαιτείται για κάθε επανάληψη του βρόχου μπορεί να ποικίλλει σημαντικά. Ξεκινώντας με μεγαλύτερα κομμάτια και μειώνοντας το μέγεθός τους καθώς προχωρά ο βρόχος, ο προγραμματιστής μπορεί να βοηθήσει στην πιο ομοιόμορφη εξισορρόπηση του φόρτου εργασίας στα νήματα.

4) Ο ιδανικός αριθμός threads. (5%)
Ο ιδανικός αριθμός threads είναι 40. Ο HPC έχει 40 physical cores και όπως θα δούμε και από τις μετρήσεις μέχρι τα 40 threads έχουμε βελτίωση εκτέλεσης προγράμματος. Αν χρησιμοποιήσουμε περισσότερα από 40 threads έχουμε χειρότερους χρόνους λόγω των overhead που προκύπτουν.

Ο βέλτιστος αριθμός threads είναι στα 8-10 όπου όπως θα δούμε από τις γραφικές έχουμε τον καλύτερο συνδυασμό speedup/efficiency.

5) Αναμενόμενη επιτάχυνση (Speedup) των πιο πάνω 3 μεθόδων, η καταμετρημένη, efficiency (10%)

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Number of Threads	Ideal Speed up	Serial -O0	Serial -O3	OMP - Static -O0	OMP - Static -O0 Speedup	OMP-Static -O3	OMP-Static -O3 Speedup	OMP - Dynamic -O0	OMP - Dynamic -O0 Speedup	OMP-Dynamic -O3	OMP-Dynamic -O3 Speedup	OMP - Guided -O0	OMP - Guided -O0 Speedup	OMP- Guided -O3	OMP- Guided -O3 Speedup
2	1	1	9,016654	5,1442	9,045619	0,996797897	5,17897	0,99328631	9,022585	0,99934265	5,144206	0,999998834	9,04247	0,997145028	5,149878	0,99889745
3	2	2	9,073878	5,145836	6,277828	1,445384932	3,745576	1,373843703	6,701362	1,354034896	3,974107	1,294840828	6,561717	1,382851165	3,792364	1,356894011
4	3	3	9,045318	5,164883	5,057119	1,788630641	2,867231	1,801348758	4,585379	1,972643483	2,749012	1,878814279	4,732914	1,911151988	2,931896	1,761618761
5	4	4	9,054823	5,169387	3,604913	2,511800701	2,077706	2,488026217	3,381125	2,678050353	1,744119	2,963895812	3,364518	2,691269002	1,754435	2,944468236
6	5	5	9,203788	5,165197	3,38364	2,720084879	1,786874	2,890633027	3,54354	2,59734277	1,802218	2,866022313	3,112965	2,956598613	1,576608	3,27614537
7	6	6	9,066298	5,206579	2,927166	3,097295473	1,647814	3,159688533	2,961538	3,061347854	1,524235	3,415863696	2,957154	3,065886322	1,516857	3,432478474
8	7	7	9,068557	5,171035	3,127486	2,899631525	1,594573	3,242896374	2,862142	3,168451111	1,441909	3,586242266	3,093624	2,931370134	1,537333	3,363640148
9	8	8	9,063142	5,204357	2,606052	3,477728764	1,382952	3,763223163	2,485902	3,645816287	1,212281	4,293028597	2,570621	3,525662476	1,2373	4,206220803
10	9	9	9,134186	5,158044	2,679454	3,408972873	1,489549	3,46282264	2,474661	3,691085766	1,257939	4,100392785	2,354198	3,879956571	1,135765	4,541471167
11	10	10	9,032985	5,150212	2,372463	3,807429241	1,210872	4,25330836	2,278708	3,964081839	1,083803	4,751981679	2,215809	4,076608137	0,999479	5,152866659
12	11	11	9,091213	5,186411	2,482761	3,66173506	1,305712	3,972094152	2,459489	3,698382867	1,077814	4,811972196	2,437261	3,730094151	1,121469	4,624658372
13	12	12	9,315071	5,252018	2,404113	3,87463942	1,186243	4,42743856	2,464699	3,779394969	1,159364	4,530085461	2,440608	3,816701002	1,143034	4,594804704
14	13	13	9,178742	5,197624	2,395893	3,831031686	1,269903	4,092929932	2,22014	4,134307746	1,105226	4,702770293	2,24545	4,087707141	1,032278	5,03510101
15	14	14	9,092761	5,1791	2,216928	4,101513897	1,171512	4,420868075	2,100177	4,329521274	0,921448	5,62061017	2,162818	4,204126746	0,962245	5,382309079
16	15	15	9,137194	5,203888	2,356787	3,876970638	1,076663	4,833489968	2,180524	4,190366169	1,082937	4,805346941	2,150175	4,249511784	0,98214	5,298519559
17	16	16	9,152113	5,301461	2,256123	4,056566508	1,080567	4,906184438	2,262904	4,044410633	1,202667	4,408087193	2,314752	3,953820107	1,090214	4,862770979
18	17	17	9,140943	5,237973	2,338627	3,908679323	1,175091	4,457504142	2,404563	3,801498651	1,111243	4,713616194	2,374849	3,849062825	1,130538	4,633168456
19	18	18	9,112968	5,315614	1,997795	4,561513068	0,937839	5,66793874	1,950026	4,673254613	0,846995	6,275850507	1,955812	4,659429434	0,864625	6,147883765
20	19	19	9,106886	5,183661	2,326446	3,914505645	1,050929	4,932455951	2,325367	3,916322026	1,13155	4,58102691	2,238064	4,069090964	0,988997	5,241331369
21	20	20	9,012181	5,20726	1,924296	4,683365241	0,924541	5,632265092	1,832674	4,917503604	0,898833	5,793356497	1,82885	4,927785767	0,786923	6,617242064
22	21	21	9,038017	5,14879	2,215013	4,080344901	1,056746	4,872306117	2,193472	4,120415943	1,101551	4,674127662	2,21977	4,071600861	1,025194	5,022259202
23	22	22	9,028512	5,147219	1,895741	4,762523995	0,925272	5,56292528	1,876297	4,811877864	0,851305	6,046268964	1,895184	4,763923714	0,830915	6,194639644
24	23	23	9,022111	5,149026	2,152364	4,191721753	1,003804	5,129513331	2,068116	4,362478217	0,93955	5,480310787	2,127958	4,239797496	0,965932	5,330629899
25	24	24	9,259022	5,152085	2,035635	4,54846866	0,903045	5,705236173	2,07116	4,470452307	1,104884	4,663009873	2,122096	4,363149452	0,912445	5,646460883
26	25	25	9,109477	5,202365	1,88341	4,836693551	0,88067	5,90728082	1,826213	4,988178816	1,065109	4,884349865	1,884896	4,852880435	0,825619	6,301169183
27	26	26	9,195263	5,250083	2,023524	4,544182822	0,971219	5,40560162	2,016379	4,560288615	1,008175	5,207435779	2,032318	4,524519785	0,880554	5,962452613
28	27	27	9,012165	5,148888	2,012003	4,479200578	0,97863	5,261322461	1,993857	4,519965574	0,895394	5,750416018	2,023052	4,454737199	0,890955	5,779066283
29	28	28	9,020732	5,149806	1,961685	4,598461017	0,889373	5,790378165	1,857965	4,855167885	0,783866	6,569752993	1,82539	4,941810791	0,743152	6,929680604
30	29	29	9,147367	5,257253	2,013632	4,542720318	0,935178	5,621660261	1,918841	4,767131305	0,871273	6,033990494	2,001072	4,571233319	0,833343	6,30863042
31	30	30	9,045887	5,170654	1,86706	4,844989984	0,935148	5,529236014	1,905474	4,747315891	0,811384	6,372634905	1,954557	4,628100894	0,803865	6,432241732
32	31	31	9,058386	5,184196	1,946386	4,653951477	0,944808	5,48703652	1,908389	4,746614029	1,007797	5,144087549	2,012982	4,499983606	0,867959	5,972858165
33	32	32	9,27182	5,158038	1,901527	4,87598651	0,963155	5,355356095	1,897791	4,885585399	0,845191	6,102807531	1,992886	4,652458796	0,828198	6,228025182
34	33	33	9,135945	5,177467	1,706808	5,352649507	0,811009	6,383982175	1,756076	5,202477	0,736553	7,029320361	1,726988	5,290103348	0,731165	7,081119857
35	34	34	9,056052	5,207231	1,922671	4,710141257	0,924868	5,63024237	1,834634	4,936162744	0,818938	6,358516762	1,973383	4,589100038	0,909807	5,723445742
36	35	35	9,069487	5,213998	1,9648	4,615984833	1,041895	5,004341128	1,850492	4,901121972	0,871404	5,983445107	1,939921	4,67518368	0,856357	6,08857988
37	36	36	9,062943	5,234099	1,880891	4,818430733	0,852075	6,142767949	1,847711	4,904956998	0,870294	6,014173371	1,933049	4,688418659	0,838849	6,239620003
38	37	37	9,123098	5,149899	1,914225	4,765948622	0,844378	6,096044504	1,836596	4,967395116	0,77084	6,680892273	1,844146	4,947058422	0,798458	6,4904494
39	38	38	9,030259	5,160224	1,883399	4,794660611	0,877305	5,881904241	1,922091	4,698143324	0,826327	6,244772348	1,860527	4,85360277	0,803939	6,418676044
40	39	39	9,036041	5,149749	1,894894	4,768626108	0,830048	6,204158073	1,851546	4,880268165	0,812415	6,338815753	2,024894	4,46247606	0,840703	6,125527089
41	40	40	9,049671	5,156846	1,777047	5,092533287	0,733921	7,026432	1,765799	5,124972521	0,822692	6,268258352	1,861097	4,862546659	0,850291	6,064801344

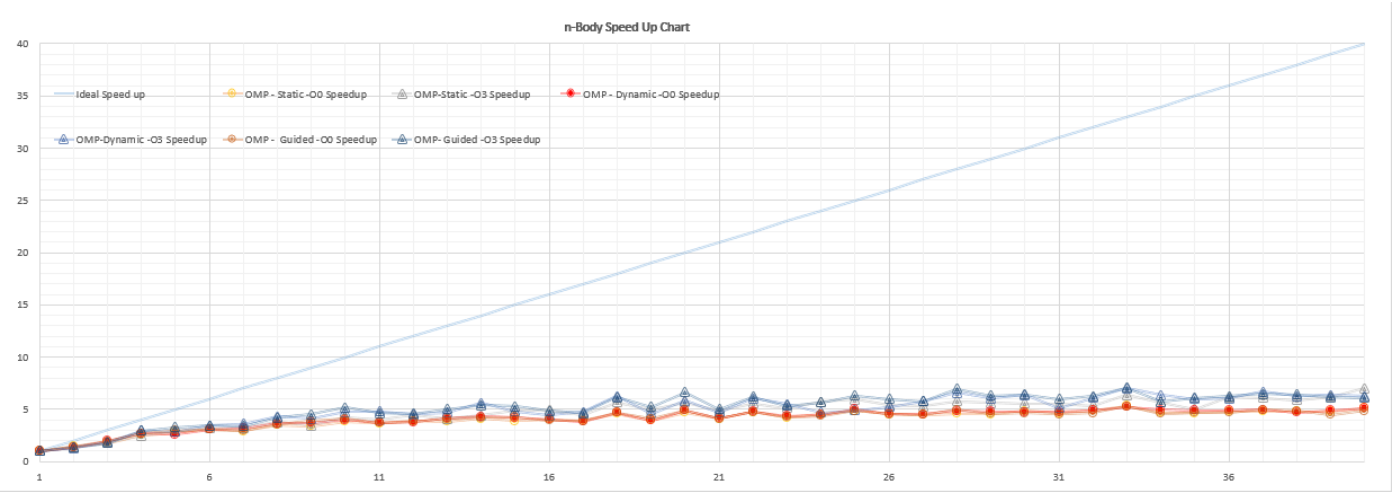
Threads	Efficiency O0 Static	Efficiency O0 Dynamic	Efficiency O0 Guided	Efficiency O3 Static	Efficiency O3 Dynamic	Efficiency O3 Guided
1	**0,996797897**	0,99934265	0,997145028	0,99328631	**0,999998834**	0,99889745
2	**0,722692466**	0,677017448	0,691425583	0,686921851	**0,647420414**	0,678447006
3	**0,596210214**	0,657547828	0,637050663	0,600449586	**0,626271426**	0,587206254
4	**0,627950175**	0,669512588	0,67281725	0,622006554	**0,740973953**	0,736617059
5	**0,544016976**	0,519468554	0,591319723	0,578126605	**0,573204463**	0,655229074
6	**0,516215912**	0,510224642	0,510981054	0,526614755	**0,569310616**	0,572079746
7	**0,414233075**	0,452635873	0,418767162	0,463270911	**0,512320324**	0,480520021
8	**0,434716095**	0,455727036	0,44070781	0,470402895	**0,536628575**	0,5257776
9	**0,378774764**	0,410120641	0,431106286	0,384758071	**0,455599198**	0,504607907
10	**0,380742924**	0,396408184	0,407660814	0,425330836	**0,475198168**	0,515289666
11	**0,332885005**	0,336034806	0,339099468	0,361099468	**0,437452018**	0,420423488
12	**0,322886618**	0,314949581	0,318058417	0,368953213	**0,377507122**	0,382900392
13	**0,294694745**	0,318023673	0,314439011	0,314840764	**0,361751561**	0,387315462
14	**0,292965278**	0,30925152	0,300294768	0,315776291	**0,401472155**	0,384450649
15	**0,258464709**	0,279357745	0,283300786	0,322223265	**0,320356463**	0,353234637
16	**0,253535407**	0,252775665	0,247113757	0,306636527	**0,27550545**	0,303923186
17	**0,229922313**	0,223617568	0,22641546	0,262206126	**0,277271541**	0,272539321
18	**0,253417393**	0,259625256	0,258857191	0,314885486	**0,348658361**	0,341549098
19	**0,206026613**	0,206122212	0,214162682	0,259602945	**0,241106679**	0,275859546
20	**0,234168262**	0,24587518	0,246389288	0,281613255	**0,289667825**	0,330862105
21	**0,194302138**	0,196210283	0,193885746	0,232014577	**0,222577508**	0,2391552
22	**0,216478363**	0,218721721	0,216541987	0,25286024	**0,274830407**	0,281574529
23	**0,182248772**	0,189672966	0,184339022	0,223022319	**0,238274382**	0,231766517
24	**0,189519527**	0,186268846	0,181797894	0,237718174	**0,194292078**	0,235269203
25	**0,193467742**	0,199527153	0,193315217	0,236291233	**0,195373995**	0,252046767
26	**0,174776262**	0,175396101	0,174019992	0,207907755	**0,200286618**	0,2293251
27	**0,165896318**	0,167406132	0,164990267	0,194863795	**0,212978371**	0,214039492
28	**0,164230751**	0,173398853	0,176493243	0,20679922	**0,234634035**	0,247488593
29	**0,156645528**	0,164383838	0,157628735	0,193850354	**0,208068638**	0,21753898
30	**0,161499666**	0,158243863	0,15427003	0,184307867	**0,212421164**	0,214408058
31	**0,150127467**	0,153116582	0,145160761	0,177001178	**0,165938308**	0,192672844
32	**0,152374578**	0,152674544	0,145389337	0,167354878	**0,190712735**	0,194625787
33	**0,1622015**	0,157650818	0,160306162	0,193454005	**0,213009708**	0,21457939
34	**0,138533566**	0,145181257	0,134973531	0,165595364	**0,187015199**	0,168336639
35	**0,131885281**	0,140032056	0,133576677	0,142981175	**0,170955574**	0,173959425
36	**0,133845298**	0,136248806	0,130233852	0,170632443	**0,167060371**	0,173322778
37	**0,128809422**	0,134253922	0,133704282	0,164839041	**0,180564656**	0,175417551
38	**0,126175279**	0,123635351	0,127726389	0,154786954	**0,164336114**	0,168912527
39	**0,122272464**	0,125135081	0,114422463	0,159080976	**0,162533737**	0,157064797
40	**0,127313332**	0,128124308	0,121563666	0,1756608	**0,156706459**	0,151620034

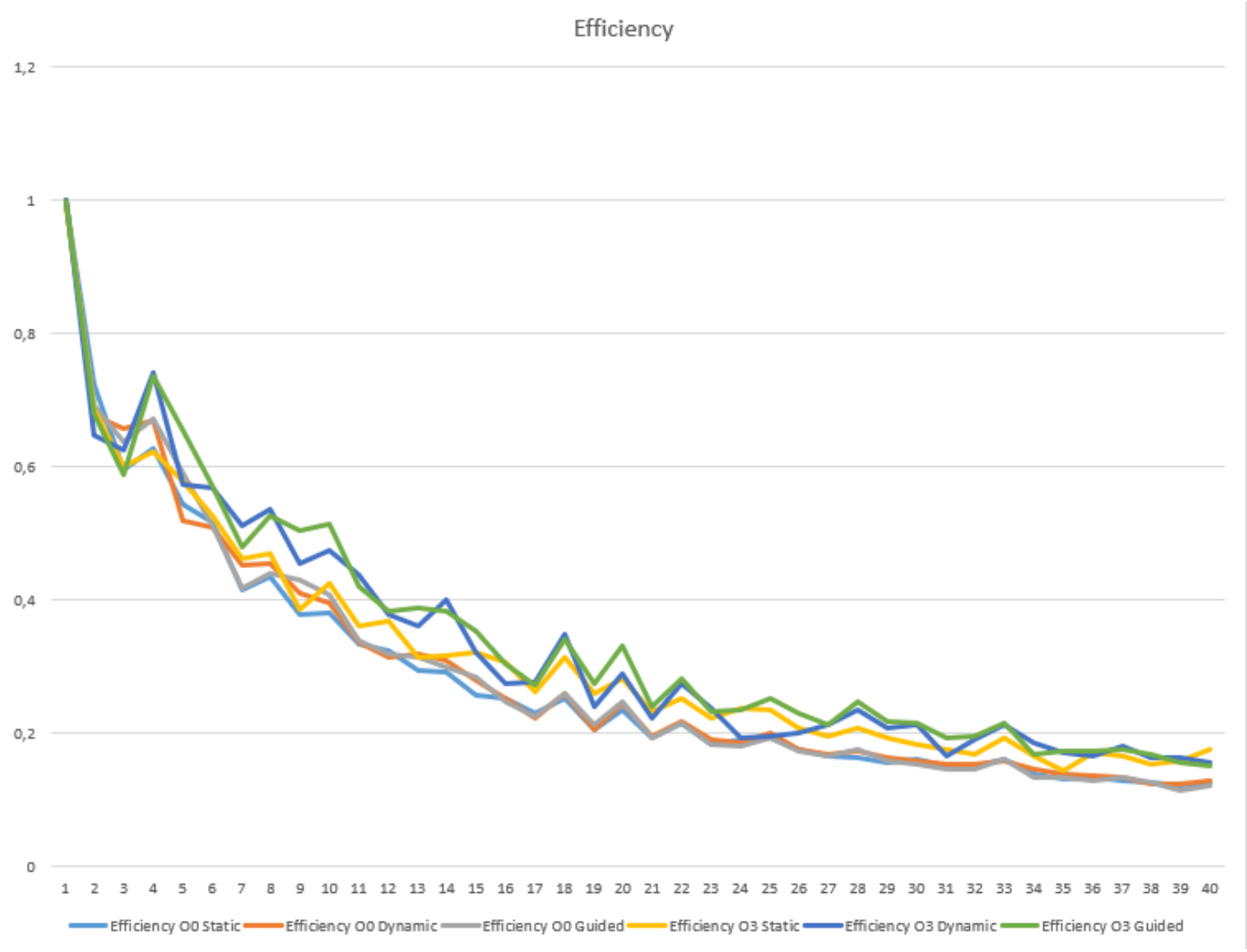
Τα αποτελέσματα δείχνουν ότι για έως και 4 νήματα, η επιτάχυνση αυξάνεται γραμμικά με την αύξηση του αριθμού των νημάτων. Ωστόσο, για μεγαλύτερες τιμές νημάτων, η επιτάχυνση δεν αυξάνεται τόσο πολύ και παραμένει σχεδόν σταθερή. Οι πραγματικές τιμές επιτάχυνσης δεν είναι ιδανικές επειδή ορισμένα μέρη του κώδικα δεν ήταν παραλληλισμένα, (resolveCollusion() 10-15% δεν ήταν)επομένως βάση του νόμου του Amdahl η επιτάχυνση μας είχε οροφή στα x5 – x10/

Να σημειωθεί επίσης ότι δεν υπάρχουν σημαντικές διαφορές στο είδος scheduling, αλλά όταν ο κώδικας συντάσσεται με τη σημαία βελτιστοποίησης O3, ο παραλληλισμός αξιοποιείται καλύτερα. Επιπλέον, το να αναφερθεί η χρήση του νόμου του Amdahl για τον υπολογισμό της θεωρητικής επιτάχυνσης για κάθε αριθμό νημάτων. Το γράφημα δείχνει ότι οι πραγματικές τιμές επιτάχυνσης είναι σχεδόν οι ίδιες με τις τιμές που λαμβάνονται μέσω του μαθηματικού τύπου, υποδεικνύοντας ότι η μέγιστη δυνατή επιτάχυνση εξήχθη από το πρόγραμμα αφού τελικά παραλληλίστηκε το 85-90% του κώδικα.

Συνοπτικά, υπάρχουν τα πλεονεκτήματα αλλά και οι περιορισμοί της παραλληλοποίησης κώδικα χρησιμοποιώντας πολλαπλά νήματα. Φένεται ότι η αύξηση του αριθμού των νημάτων αρχικά οδηγεί σε γραμμική αύξηση της ταχύτητας, αλλά γίνεται λιγότερο αποτελεσματική για μεγαλύτερους αριθμούς νημάτων. Να τονιστεί επίσης η σημασία της βελτιστοποίησης και της παραλληλοποίησης όσο το δυνατόν μεγαλύτερου μέρους του κώδικα για να επιτευχθεί η μέγιστη δυνατή επιτάχυνση.

6) Γραφική Παράσταση και Σχολιασμός (~200 λέξεις) για τις λύσεις πιο πάνω. (15%-25%)





---

7) Bonus: `resolveCollisions`: Κώδικας και επεξήγηση/ορθότητα λύσης (~200 λέξεις). (10%+10%)

```
void resolveCollisions_static(int number_of_threads)
{
    int *t1 = (int *)malloc(10000 * sizeof(int));
    int *t2 = (int *)malloc(10000 * sizeof(int));

    int i, j, c = 0;
    double dx, dy, dz, md;
#pragma omp parallel for private(j, dx, dy, dz, md) schedule(static, bodies / number_of_threads) num_threads(number_of_threads)
    for (i = 0; i < bodies - 1; i++)
    {
        for (j = i + 1; j < bodies; j++)
        {
            md = masses[i] + masses[j];
            dx = fabs(positions[i].x - positions[j].x);
            dy = fabs(positions[i].y - positions[j].y);
            dz = fabs(positions[i].z - positions[j].z);
            // if(positions[i].x==positions[j].x && positions[i].y==positions[j].y && positions[i].z==positions[j].z){

            if (dx < md && dy < md && dz < md)
            {
#pragma omp critical
                {

                    *(t1 + cnt) = i;
                    *(t2 + cnt) = j;
                    c++;
                }
            }
        }
    }
    // printf(" %d",cnt);
    bubblesort(t1, t2, c);
    for (i = 0; i < cnt; i++)
    {
        // Swap Velocities
        int temp = velocities[t1[i]];
        velocities[t1[i]] = velocities[t2[i]];
        velocities[t2[i]] = temp;
    }
}
```

Θα επεξηγήσω για το schedule static αλλά ανάλογα παραλληλίζεται και για dynamic και για guided. Οι διαφορές μεταξύ των τριών συζητήθηκαν στα προηγούμενα υποερωτήματα.

Το parallel for directive δημιουργεί μια ομάδα νημάτων για την παράλληλη εκτέλεση του βρόχου for. Οι κοινόχρηστες μεταβλητές στον βρόχο, δηλ. μάζες, θέσεις και ταχύτητες πινάκων, καθορίζονται στη shared cluase. Η private clauseκαθορίζει ότι κάθε νήμα έχει το αντίγραφό του από τις μεταβλητές ευρετηρίου βρόχου i, j, dx, dy, dz και md, οι οποίες δεν μοιράζονται μεταξύ των νημάτων.

Η στρατηγική static scheduling διαιρεί τις επαναλήψεις του βρόχου σε κομμάτια περίπου ίσου μεγέθους, με κάθε νήμα να εκτελεί ένα κομμάτι επαναλήψεων. Αυτό προσδιορίζεται στην ρήτρα χρονοδιαγράμματος ως "static, body / number_of_threads", το οποίο διασφαλίζει ότι οι επαναλήψεις κατανέμονται εξίσου μεταξύ των νημάτων.

Επιπλέον, χρησιμοποιήσαμε την κρίσιμη οδηγία για να διασφαλίσουμε ότι η κοινόχρηστη μεταβλητή cnt ενημερώνεται με ασφάλεια μόνο από ένα νήμα τη φορά. Αυτό διασφαλίζει ότι η μεταβλητή cnt αυξάνεται μόνο κατά ένα νήμα τη φορά, αποφεύγοντας τις συνθήκες αγώνα.

Τέλος, ταξινομούμε τους δείκτες των σωμάτων που συγκρούονται χρησιμοποιώντας τον αλγόριθμο ταξινόμησης με φυσαλίδες και ανταλλάσσουμε τις ταχύτητες τους.

Ο λόγος που πρέπει να ταξινομήσουμε τα ζεύγη των σωμάτων που συγκρούονται πριν επιλύσουμε τις συγκρούσεις είναι ότι θέλουμε να αποφύγουμε την πιθανότητα αδιεξόδου, όπου δύο νήματα ταυτόχρονα προσπαθούν να ανταλλάξουν τις ταχύτητες των ίδιων δύο σωμάτων. Εάν συνέβαινε αυτό, και τα δύο νήματα θα περίμεναν το ένα το άλλο για να απελευθερώσει τους πόρους που χρειάζονται και το πρόγραμμα θα κρεμόταν επ' αόριστον.

Ταξινομώντας τα ζεύγη των σωμάτων που συγκρούονται με σταθερή σειρά, μπορούμε να διασφαλίσουμε ότι κάθε ζεύγος χειρίζεται μόνο ένα νήμα και ότι δεν υπάρχουν αντικρουόμενες πράξεις στο ίδιο ζεύγος σωμάτων. Επομένως, η ταξινόμηση είναι απαραίτητη για την αποφυγή συνθηκών αγώνα και τη διασφάλιση της σωστής εκτέλεσης του προγράμματος.