

Παράλληλη Επεξεργασία



Απαντήσεις Project

Χρήστος Λυκούδης (1067405) — Λουδάρος Ιωάννης (1067400)
Τσίκελης Ιωάννης (1067407) — Χριστίνα Κρατημένου (1067495)

Γενικές Πληροφορίες

Στις επόμενες σελίδες αναλύονται οι απαντήσεις τις ομάδας μας στο Project του μαθήματος “Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών”. Σε αυτή την σελίδα έχετε πρόσβαση σε γενικές πληροφορίες γύρω από το Project αλλά και γύρω από τα μέλη της ομάδας μας.



Για να έχετε πρόσβαση στην τελευταία έκδοση των απαντήσεων μπορείτε να σκανάρετε το παραπάνω QR Code ή να χρησιμοποιήσετε το παρακάτω κουμπί.

Πατήστε Εδώ

Για την υλοποίηση του Project εργαστήκαμε σε μια ομάδα 4 ατόμων. Κάναμε συναντήσεις σε εβδομαδιαία βάση για να αξιολογήσουμε την πρόοδο μας αλλά και να καθορίσουμε τους επόμενους μας στόχους. Όλοι είχαμε επίγνωση των εργασιών που πραγματοποιούν οι συνεργάτες μας και δίναμε feedback ο ένας στον άλλον με αποτέλεσμα να προχωράμε συνεκτικά. Ενώ η σύνταξη του κώδικα γινόταν από όλα τα μέλη της ομάδας, το compilation και η εκτέλεση του γινόταν από το ίδιο μηχάνημα ώστε να υπάρχει συνοχή στα αποτέλεσματα. Η ομάδα αποτελείται από τα εξής άτομα:

Ιωάννης Λουδάρος (1067400)

Χριστίνα Κρατημένου (1067495)

Χρήστος Λυκούδης (1067405)

Ιωάννης Τσικέλης (1067407)

Παρακάτω υπάρχουν αναλυτικότερες πληροφορίες για τα μέλη της ομάδας μας.



Ιωάννης Τσικέλης
1067407

st1067407@ceid.upatras.gr
Φοιτητής 3ου έτους



Χρήστος Λυκούδης
1067405

up1067405@upnet.gr
Φοιτητής 3ου έτους



Ιωάννης Λουδάρος
1067400

iloudaros@upnet.gr
Φοιτητής 3ου έτους



Χριστίνα Κρατημένου
1067495

up1067495@upnet.gr
Φοιτήτρια 3ου έτους

“Τα Bio ήταν ιδέα του Γιάννη του Λουδάρου, πάρτε τον, δεν τον αντέχω”



Περιεχόμενα

1. Εξερευνώντας το Πρόβλημα	3
2. OpenMP	4
3. OpenMP Tasks	6
4. MPI	7
5. MPI - OpenMP Hybrid	9
6. Συμπεράσματα	10

Εξερευνώντας το Πρόβλημα.

Στόχοι Παραλληλοποίησης

Έπειτα από την μελέτη του αρχικού κώδικα που βρέθηκε στο “[multistart_hooke_seq.c](#)” αντιληφθήκαμε ότι προοπτική παραλληλοποίησης εντοπίζεται κυρίως στο εξής σημείο του κώδικα:

```
for ( trial = 0; trial < ntrials; trial++ ) {...}
```

Τα `ntrials` βλέπουμε ότι φτάνουν αυτό το for-loop στις 131.072 επαναλήψεις, φανερώνοντας μας ότι εδώ θα έχουμε μεγάλο προσδοκόμενο κέρδος σε χρόνο από την εφαρμογή παραλληλοποίησης.

Άλλο ένα σημείο το οποίο θα αφελούσε να παραλληλοποιήσουμε (το οποίο όμως θα είχε μικρότερο αντίκτυπο στον χρόνο εκτέλεσης του προγράμματος) είναι το:

```
for ( i = 0; i < MAXVARS; i++ ) best_pt[ i ] = 0.0;
```

Μιας και είναι ένα for loop που εκτελείται 250 φορές που μπορεί να παραλληλοποιηθεί άριστα, αφού οι επαναλήψεις δεν σχετίζονται μεταξύ τους.

Δυνατότητες

Η αναφορά αυτή συντάχθηκε σύμφωνα με τα συμπεράσματα που προέκυψαν από το compilation και την εκτέλεση του κώδικα σε περιβάλλον με τα εξής χαρακτηριστικά:



MacBook Pro 13”
Dual-Core Intel Core i5 2,4 GHz (Hyper-Threading Enabled)
8 GB 1600 MHz

Μπορούμε συνεπώς εύκολα να συμπεράνουμε πριν ακόμη ξεκινήσουμε ότι αν και η αύξηση των Threads (OpenMP) θα έχει θετικά αποτελέσματα στον χρόνο εκτέλεσης, η αύξηση των ranks (MPI) θα εξαντλήσει τους πόρους του μηχανήματος μας και θα καταστήσει την εκτέλεση (και το compilation όπως προέκυψε) αδύνατη.

Αρχική Επίδοση

Παραθέτουμε το χρόνο εκτέλεσης του δισμένου κώδικα “[multistart_hooke_seq.c](#)”.

```
➔ 0.Original Code git:(main) ✘ ./multistart_hooke_seq

FINAL RESULTS:
Elapsed time = 92.780 s
Total number of trials = 131072
Total number of function evaluations = 3133429415
```

OpenMP.

Προετοιμασία

Η παραλληλοποίηση με χρήση OpenMP χρειάστηκε την εξής προετοιμασία:

- i. Εγκατάσταση του gcc μέσω HomeBrew.

```
$brew install gcc
```

- ii. Απόφαση σχετικά με το πλήθος των νημάτων: Ακόμη και με την αξιοποίηση του Hyper-Threading δεν θα είχε νόημα η χρήση περισσότερων από 4 Threads, αφού οι φυσικοί πυρήνες του συστήματος είναι μόλις 2.

Υλοποίηση

Κομβικά σημεία στην υλοποίηση του κώδικα με χρήση OpenMP είναι τα παρακάτω:

- i. Εισαγωγή του “`omp.h`” στον κώδικα.

```
136| #include <omp.h>
```

- ii. Δήλωση έναρξης παράλληλου τμήματος.

```
327| #pragma omp parallel num_threads(4) // Split to 4 threads.
```

- iii. Χρήση της `thread-safe erand48()` αντί της `srand(48)`.

```
335| // Variables used by erand() -- rand() safe for multithreading.
```

```
336| short seed = (short)get_wtime(); // Seed for erand().
```

```
337| unsigned short randBuffer[3];
```

```
338| randBuffer[0] = 0;
```

```
339| randBuffer[1] = 0;
```

```
340| randBuffer[2] = seed + omp_get_thread_num();
```

...

```
348| startpt[i] = 4.0 * erand48(randBuffer) - 4.0;
```

- iv. Παραλληλοποίηση του for-loop.

```
342| #pragma omp for // Parallelized for-loop using OpenMP.
```

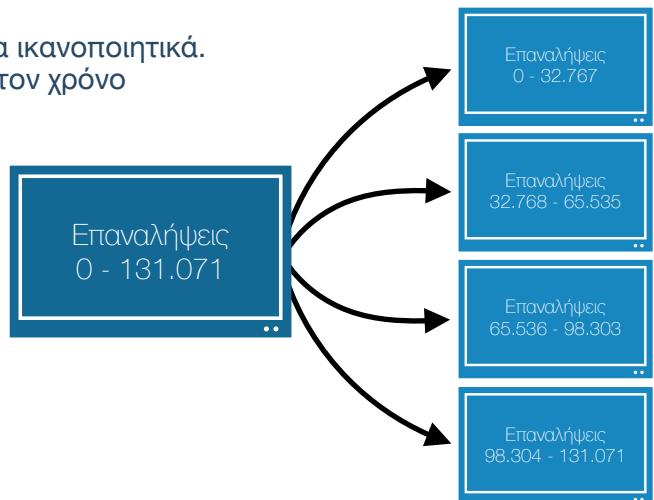
- v. Ορισμός Critical τμήματος λόγω του race condition που προκύπτει στο σημείου που γίνεται η σύγκριση για τον ορισμό των `best_trial`, `best_jj` και `best_fx`.

```
365|     if (fx < best_fx)
366|     {
367| #pragma omp critical // Only one thread can access the shared arrays each time.
368|     {
369|         best_trial = trial;
370|         best_jj = jj;
371|         best_fx = fx;
372|         for (int i = 0; i < nvars; i++)
373|             best_pt[i] = endpt[i];
374|     }
375| }
```

Αποτελέσματα

Τα αποτελέσματα της υλοποίησης αυτής ήταν ιδιαίτερα ικανοποιητικά.
Όπως μας γίνεται φανερό, καταφέραμε να μειώσουμε τον χρόνο
εκτέλεσης σχεδόν στο μισό.

```
+ 1.OpenMP git:(main) ✘ ./multistart_hooke_omp
FINAL RESULTS:
Elapsed time = 50.523 s
Total number of trials = 131072
Total number of function evaluations = 1158779806
```



OpenMP Tasks.

Προετοιμασία

Η απαιτούμενη προετοιμασία έχει γίνει ήδη από το προηγούμενο ζητούμενο.

Υλοποίηση

Κομβικά σημεία στην υλοποίηση του κώδικα με χρήση OpenMP είναι τα παρακάτω (τα σημεία i, ii, iii και ν είναι όμοια με το προηγούμενο ζητούμενο, για αυτό δεν αναλύονται) :

- i. Εισαγωγή του “`omp.h`” στον κώδικα.
- ii. Δήλωση έναρξης παράλληλου τμήματος.
- iii. Χρήση της thread-safe `erand48()` αντί της `srand(48)`.
- iv. Ορισμός του Task που θα διαμοιραστεί στα Threads.

```
342| // One thread enters the for-loop and distributes the generated tasks (in our case each task in  
one whole repetition) to all threads available  
343| #pragma omp single nowait  
344|     for (trial = 0; trial < ntrials; trial++)  
345|     {  
346| #pragma omp task  
347|     {...}
```

- v. Ορισμός Critical τμήματος λόγω του race condition που προκύπτει στο σημείου που γίνεται η σύγκριση για τον ορισμό των `best_trial`, `best_jj` και `best_fx`.

Αποτελέσματα

Τα αποτελέσματα και αυτής υλοποίησης αυτής ήταν ιδιαίτερα ικανοποιητικά. Όπως μας γίνεται φανερό, καταφέραμε να μειώσουμε τον χρόνο εκτέλεσης σχεδόν στο μισό.

```
→ 2.OpenMP_Tasks git:(main) ./multistart_hooke_omp_tasks  
  
FINAL RESULTS:  
Elapsed time = 55.839 s  
Total number of trials = 131072  
Total number of function evaluations = 1832648704
```

MPI.

Προετοιμασία

Η παραλληλοποίηση με χρήση MPI χρειάστηκε την εξής προετοιμασία:

- i. Εγκατάσταση του open-mpi μέσω HomeBrew.

```
$brew install open-mpi
```

- ii. Απόφαση σχετικά με το πλήθος των ranks: Αν χρησιμοποιήσουμε περισσότερα από 2 ranks, το compilation είναι αδύνατο λόγω έλλειψης “slots”
- iii. Απόφαση σε σχέση με τον ρόλο του κάθε rank:
 - iii.i. Το Main Rank θα αναλάβει τον ρόλο του αξιολογητή: Δηλαδή θα περιμένει να του σταλθούν αποτελέσματα από το(a) Secondary Rank(s) (στην περίπτωση μας είναι μόνο ένα) για να υπολογίζει τα best_trial, best_jj και best_fx.
 - iii.ii. Τα Secondary Ranks αναλαμβάνουν το βάρος του υπολογισμού της hooke() και της f()

Υλοποίηση

Κομβικά σημεία στην υλοποίηση του κώδικα με χρήση MPI είναι τα παρακάτω:

- i. Εισαγωγή του “mpi.h” στον κώδικα.

```
136| #include <mpi.h>
```

- ii. Αρχικοποίηση και συλλογή πληροφοριών για το περιβάλλον εκτέλεσης.

```
299| //Initialize MPI
300| MPI_Init(&argc, &argv);
301|
302| //Get each rank (0: main, non-zero: secondaries)
303| int rank;
304| MPI_Comm_rank(MPI_COMM_WORLD, &rank);
305|
306| // Get total number of ranks to split the work between them.
307| int world_size;
308| MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

- iii. Διακριτοποίηση των ranks και ανάθεση των ρόλων τους όπως περιγράφηκαν παραπάνω.

```
347| if (rank != 0)
348| {
349|   /*****
350|   /* Secondary rank(s) only: Perform the comparisons and find the best values. */
351|   *****/
352|
353|   for (trial = 0; trial < mpi_ntrials; trial++) {...}
...
383| else if (rank == 0)
384| {
385|
386|   /*****
387|   /* Main rank only: Perform the comparisons and find the best values. */
388|   *****/
389|
390|   for (int i = 0; i < ntrials; i++) {...}
```

iv. Αποστολή μηνυμάτων από τα Secondary Ranks και παραλαβή τους από το Main Rank.

... inside the Secondary Threads

```
373| // Send the results to main rank (send fx and block until main acknowledges you).
373| MPI_Ssend(&fx, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD); //MPI_send fx (MUST BE
FIRST AND BLOCKING)
373| MPI_Ssend(&trial, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
373| MPI_Ssend(&jj, 1, MPI_INT, 0, 3, MPI_COMM_WORLD);
373| MPI_Ssend(&endpt, 250, MPI_DOUBLE, 0, 4, MPI_COMM_WORLD);
```

... inside the Main Thread

```
394| // Receive the values from each process (the first process to be acknowledged will be unblocked
and able to send all its calculated values).
395| MPI_Recv(&fx, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
396| MPI_Recv(&trial, 1, MPI_INT, MPI_ANY_SOURCE, 2, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
397| MPI_Recv(&jj, 1, MPI_INT, MPI_ANY_SOURCE, 3, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
398| MPI_Recv(&endpt, 250, MPI_DOUBLE, MPI_ANY_SOURCE, 4, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
```

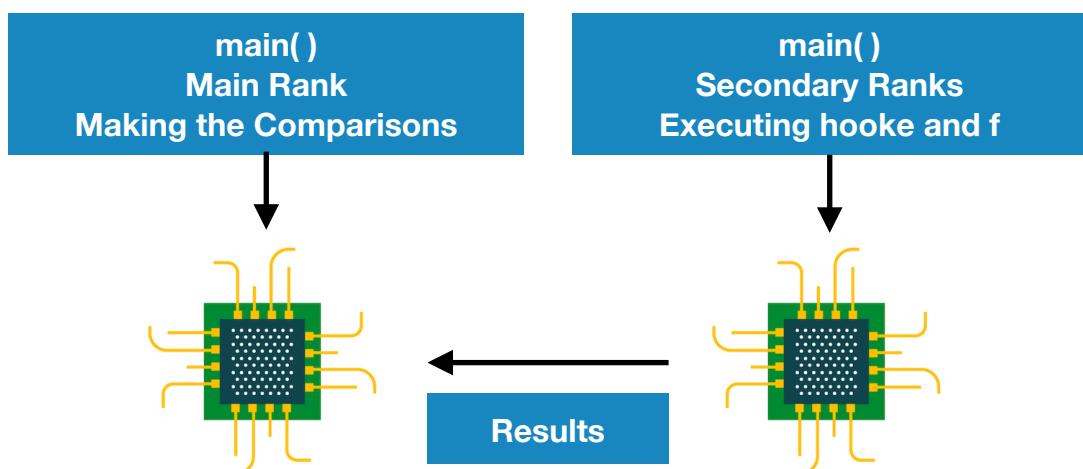
Ιδιαίτερο ενδιαφέρον έχει η χρήση του MPI_ANY_SOURCE, που απαλλάσει την υλοποίηση από περιπτούς ελέγχους.

Αποτελέσματα

Τα αποτελέσματα αυτής υλοποίησης ήταν μη ικανοποιητικά. Παρατηρώντας τις καλύτερες περιπτώσεις της σειριακής υλοποίησης, συνειδητοποιήσαμε ότι είναι καλύτερες από τη μέση περίπτωση της υλοποίησης με MPI.

```
→ 3.MPI git:(main) ✘ mpirun ./multistart_hooke_mpi

FINAL RESULTS:
Elapsed time = 92.613 s
Total number of trials = 131072
Total number of function evaluations = 0
Best result at trial 119234 used 19 iterations, and returned
```



MPI - OpenMP Hybrid.

Προετοιμασία

Η σχεδιαστική ιδέα πίσω από αυτή την υλοποίηση είναι η εξής: Να χρησιμοποιήσουμε τους ρόλους των Ranks όπως δώθηκαν στο προηγούμενο ζητούμενο, αλλά χρησιμοποιώντας το OpenMP στα Secondary Ranks ώστε να μοιράσουμε όσο περισσότερο μπορούμε το βάρος της εκτέλεσης των `hook()` και `f()`.

Υλοποίηση

Κομβικά σημεία στην υλοποίηση του κώδικα με χρήση MPI και OpenMP είναι τα παρακάτω:

- i. Εισαγωγή του “`mpi.h`” και “`omp.h`” στον κώδικα.

```
136| #include <mpi.h>
137| #include <omp.h>
```

- ii. Αρχικοποίηση και συλλογή πληροφοριών για το περιβάλλον εκτέλεσης όπως στο προηγούμενο ζητούμενο.
- iii. Διακριτοποίηση των ranks και ανάθεση των ρόλων τους όπως στο προηγούμενο ζητούμενο.
- iv. Χρήση του OpenMP για παραλληλοποίηση του for-loop του Secondary Rank.
iv.i. Δήλωση έναρξης παράλληλου τμήματος εντός του Secondary Rank.

```
347| if (rank != 0)
348| {
349|   /*****
350|   /* Secondary rank(s) only: Perform the comparisons and find the best values. */
351|   *****/
352|
353| #pragma omp parallel // Split each secondary rank to threads.
```

- iv.ii. Χρήση της `thread-safe erand48()` αντί της `srand(48)` όπως στα προηγούμενα ζητούμενα.
- iv.iii. Παραλληλοποίηση του for-loop όπως στα προηγούμενα ζητούμενα.
- iv.iv. Αποστολή μηνυμάτων μέσα από την απαιτούμενη Κρίσιμη Περιοχή.

... inside the Secondary Threads

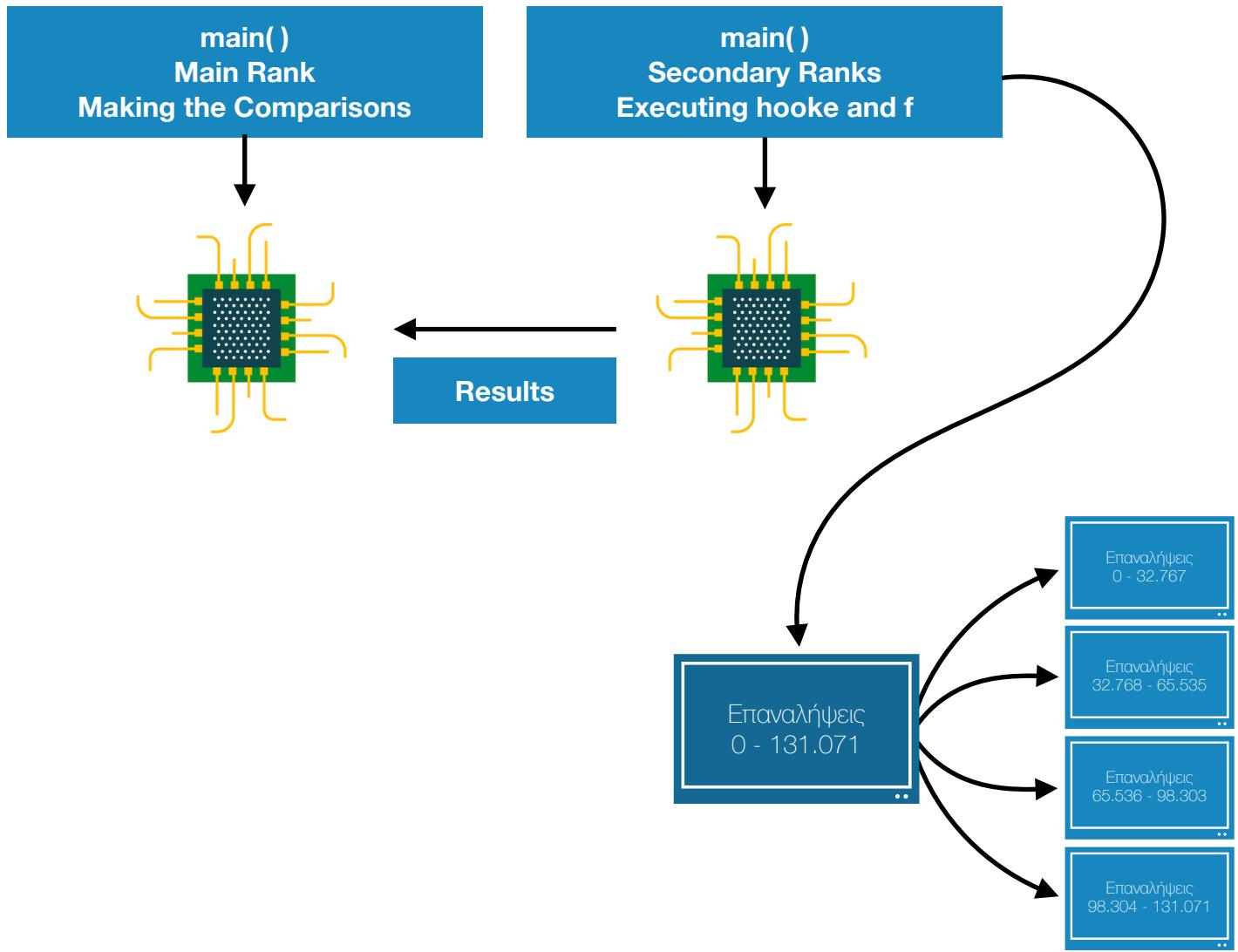
```
// Send the results to main rank, one at a time(send fx and block until main acknowledges you).
// (*) All threads can send data to the main rank (MPI_THREAD_MULTIPLE)
400| #pragma omp critical
401| // Send the results to main rank (send fx and block until main acknowledges you).
402|   MPI_Ssend(&fx, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD); //MPI_send fx (MUST BE
FIRST AND BLOCKING)
403|   MPI_Ssend(&trial, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
404|   MPI_Ssend(&jj, 1, MPI_INT, 0, 3, MPI_COMM_WORLD);
405|   MPI_Ssend(&endpt, 250, MPI_DOUBLE, 0, 4, MPI_COMM_WORLD);
```

- v. Παραλαβή μηνυμάτων από το Main Rank και αξιολόγηση των αποτελεσμάτων όπως στο προηγούμενο ζητούμενο.

Συμπεράσματα.

Τελική Μορφή

Για μια γρήγορη κατανόηση της τελικής υλοποίησης μπορείτε να συμβουλευτείτε το παρακάτω σχήμα.



Βέλτιστη υλοποίηση

Φαίνεται ότι στο μηχάνημα στο οποίο έγιναν οι υλοποιήσεις μας, οι υλοποιήσεις που δεν περιείχαν MPI ήταν αποδοτικότερες. Η παρατήρηση αυτή γίνεται φανερή από τις μετρήσεις που παρατίθενται και ήταν μάλλον αναμενόμενη, αφού τα δύο διαθέσιμα slots του μηχανήματος δεν είναι αρκετά ώστε να έχουν θετική επίδραση στον χρόνο εκτέλεσης του προγράμματος. Επιπλέον, η εναλλαγή μηνυμάτων μεταξύ διαφορετικών σημείων στην ιεραρχία της μνήμης εισάγει σημαντική καθυστέρηση που αναιρεί την όποια βελτίωση καταφέρνουμε να κερδίσουμε.