Παράλληλα Συστήματα και Προγραμματισμός - MYE023
Δεύτερο Σετ ασκήσεων - MPI

Ιωάννης Χουλιάρας

Προπτυχιακός φοιτητής στο τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Πανεπιστήμιο Ιωαννίνων

ΑΜ: 2631

cs02631@uoi.gr

Ιούνιος 2020

Πληροφορίες

Με το δεύτερο σετ ασκήσεων στο μάθημα *Παράλληλα Συστήματα και Προγραμματισμός* κλινόμαστε να παραλληλοποιήσουμε δύο σειριακά προγράμματα που μας έχει διαθέσει ο καθηγητής κύριος Δημακόπουλος στην σελίδα του μαθήματος. Η παραλληλοποίηση θα γίνει χρησιμοποιώντας το MPI. Επίσης ως άσκηση ζητείται να γράψουμε έναν μικρό οδηγό για το τι ακριβώς είναι οι *μονόπλευρες επικοινωνίες*, τι παρέχει το MPI και πώς τις χρησιμοποιεί κάποιος σε μία εφαρμογή.

Το compile και η εκτέλεση των ασκήσεων έγιναν με τις εντόλες:

- `mpicc -o name name.c`
- `mpirun -np N -hostfile nodes.txt name`

όπου nodes.txt είναι οι εικονικές μηχανές και N το πλήθος των διαθέσιμων πυρήνων (διεργασιών). Επιλέχθηκαν όλοι οι διαθέσιμοι κόμβοι με 4 πυρήνες ο κάθε ένας. Ως βασικό μηχάνημα επιλέχθηκε το παρακάτω σύστημα:

Όνομα υπολογιστή	opti3060ws07
Επεξεργαστής	Intel i3-8300 @ 3.70 GHz
Πλήθος πυρήνων	4

Contents

1 Άσκηση

1.1 Το Πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιήσουμε με χρήση MPI ένα πρόγραμμα το οποίο, δεδομένου του N, να υπολογίζει το πλήθος των πρώτων αριθμών καθώς και τον μεγαλύτερο πρώτο αριθμό μέχρι και το N. Η βασική ιδέα είναι κάθε διεργασία να αναλαμβάνει να ελέγχει ένα υποσύνολο των αριθμών.

1.2 Μέθοδος Παραλληλοποίησης

Κάθε διεργασία πρέπει να αναλαμβάνει να ελέγχει ένα υποσύνολο αριθμών. Άρα για ένα συγκεκριμένο N, κάθε διεργασία θα πρέπει να ελέγξει (N / Αριθμό Διεργασιών) αριθμούς. Για την σωστή διαμοίραση, ο αριθμός των διεργασιών θα πρέπει να διαιρεί ακριβώς τον αριθμό N. Αλλά πως μπορούμε να βρούμε πόσες διεργασίες έχουμε ή πως γίνεται η διαμοίραση;
Το MPI είναι ένα σύνολο προδιαγραφών για προγραμματιστές και χρήστες βιβλιοθηκών μεταβίβασης μηνυμάτων. Για την παραλληλοποίηση μέσω MPI χρειάζεται να εισάγουμε στο πρόγραμμα μας την βιβλιοθήκη. Επίσης γίνεται define ως COMMUNICATOR ο προκαθορισμένος communicator που περιλαμβάνει όλες τις λειτουργίες του MPI: MPI_COMM_WORLD.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define COMMUNICATOR MPI_COMM_WORLD
#define BOSS 0
```

Στην αρχή της main και για να ορίσουμε τον χώρο που θα επικοινωνούν οι διεργασίες χρησιμοποιούμε τρεις εντολές.

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(COMMUNICATOR, &procid);
MPI_Comm_size(COMMUNICATOR, &nprocs);
```

Το MPI_Comm_rank ορίζει σε κάθε διεργασία μία ταυτότητα. Με αυτό τον τρόπο μπορεί να επικοινωνεί για παράδειγμα η διεργασία 0 (BOSS) με την διεργασία 7.
Με το MPI_Comm_Size αποθηκεύεται σε μία μεταβλητή ο αριθμός διεργασιών που είναι διαθέσιμος και έχει δοθεί από τον χρήστη όπως προαναφέρθηκε.
Πλέον όλες οι διεργασίες έχουν μια πλήρη αντιγραφή όλων των μεταβλητών και τυχόν συναρτήσεων (δεν

είναι κοινή η μνήμη). Κάθε μία διεργασία θα πρέπει να πάρει συγκεκριμένη δουλειά για να εκτελέσει την συνάρτηση εύρεσης πρώτων αριθμών. Υπάρχουν διάφοροι τρόποι διαμοίρασης, όπου σε αυτή την άσκηση επιλέχθηκε ο εξής:
Κάθε διεργασία θα ξεκινάει από το id της έως την συνθήκη (N-1)/2 με ένα βήμα i + nprocs. Άρα η διεργασία 0 για nprocs = 8, θα εκτελέσει την συνάρτηση για τους αριθμούς 0, 8, 16…

```
for (i = procid; i < (UPTO - 1) / 2; i = i + nprocs)
  {
    num = 2 * i + 3;
    divisor = 1;

    do
    {
      divisor += 2;
      quotient = num / divisor;
      remainder = num % divisor;
    } while (remainder && divisor <= quotient);

    if (remainder || divisor == num)
    {
      count++;
      lastprime = num;
    }
  }
```

Όλες οι διεργασίες και αφού υπολογίσουν την μεταβλητή count και lastprime, θα πρέπει να στείλουν το αποτέλεσμα τους στην διεργασία 0 (BOSS) ώστε να εκτυπώσει στον χρήστη τα αποτελέσματα. Αυτό επιτυγχάνεται με την εντολή MPI_Reduce (λειτουργία υποβίβασης) των συλλογικών επικοινωνιών.

```
  MPI_Reduce(&count, &finalCount, 1, MPI_LONG, MPI_SUM, BOSS, COMMUNICATOR);
  MPI_Reduce(&lastprime, &last, 1, MPI_LONG, MPI_MAX, BOSS, COMMUNICATOR);
```

Για την μεταβλητή count χρησιμοποιείτε το όρισμα MPI_SUM καθώς επιθυμούμε να αποθηκευτεί στην μεταβλητή finalCount μόνο της διεργασίας BOSS η πρόσθεση όλων των counts που έχουν υπολογίσει οι διεργασίες.
Ενώ για την μεταβλητή lastprime επιθυμούμε να αποθηκευτεί στην μεταβλητή last της διεργασίας BOSS ο μεγαλύτερος πρώτος αριθμός έως το N. Άρα χρησιμοποιούμε το MPI_MAX.

Τέλος θα πρέπει να τερματίσουμε τον χώρο της επικοινωνίας καλώντας την εντολή:

```
1    MPI_Finalize();
```

1.3 Πειραματικά αποτελέσματα - μετρήσεις

Η χρονομέτρηση πραγματοποιήθηκε με την εντολή MPI_Wtime(). Κάθε πείραμα εκτελέστηκε τέσσερις φορές και υπολογίστηκαν οι μέσοι όροι. Χρονομετρήθηκε η επικοινωνία (Communication) μεταξύ διεργασιών καθώς και η υλοποίηση της συνάρτησης εύρεσης πρώτων αριθμών (Computation). Με την χρήση δύο κόμβων (άρα 8 πυρήνες) έχουμε τα παρακάτω αποτελέσματα:

Number of processes	Communication	Computation	Total
8	0,69	2,76	3,45
	0,11	2,99	3,1
	0,17	3,49	3,66
	0,47	3,09	3,56
summary:	**0,36**	**3,08**	**3,44**

Ενώ με την χρήση τεσσάρων κόμβων (16 πυρήνες) έχουμε:

Number of processes	Communication	Computation	Total
16	0	1,76	1,76
	0,64	1,05	1,69
	0	1,83	1,83
	0,68	1,37	2,05
summary:	**0,33**	**1,50**	**1,83**

Τέλος με την χρήση οχτώ κόμβων (32 πυρήνες) έχουμε τα εξής:

Number of processes	Communication	Computation	Total
32	0,26	0,86	1,12
	0,03	0,96	0,99
	0,02	1,02	1,04
	0,11	0,99	1,1
summary:	**0,10**	**0,95**	**1,05**

Επίσης ο χρόνος για την εκτέλεση του σειριακού προγράμματος είναι: **13,55** δευτερόλεπτα. Τα αποτελέσματα απεικονίζονται στο παρακάτω γράφημα 1:

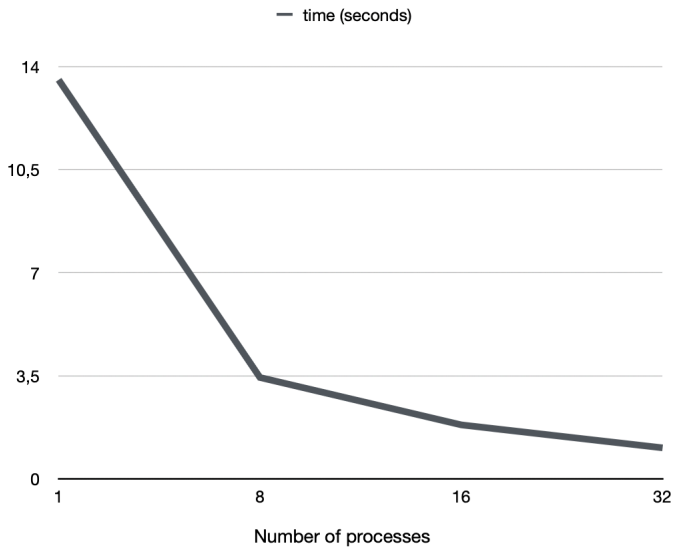

Figure 1: Χρόνος ανά αριθμό διεργασιών

1.4 Σχόλια

Παρατηρούμε ότι ο χρόνος μειώνεται υπερβολικά από το σειριακό (μονή διεργασία) με τους δύο κόμβους (8 διεργασιών). Είναι άλλωστε αναμενόμενο καθώς μειώνεται το πλήθος των επαναλήψεων. Η επικοινωνία μεταξύ των διεργασιών πραγματοποιείται σχεδόν ακαριαία και αυτό επιτυγχάνεται με την χρησιμοποίηση μόνο του MPI_Reduce. Όπως θα δούμε στην άσκηση 2, ο χρόνος της επικοινωνίας θα αυξηθεί αισθητά όταν θα επικοινωνεί στην αρχή η διεργασία 0 (BOSS) με τις υπόλοιπες.

Σε αυτή την άσκηση δεν χρειάστηκε να επικοινωνήσει η διεργασία 0 με τις υπόλοιπες ώστε να στείλει τον αριθμό N γιατί έχει γίνει define στην αρχή του προγράμματος. Σε ένα πρόγραμμα που ίσως χρειαζόταν κάποιο input από τον χρήστη και κάποια διαδικασία πριν την παραλληλοποίηση, τότε θα έπρεπε η διεργασία 0 πρώτα να στείλει τον αριθμό στις υπόλοιπες και μετά να αρχίσει η διαδικασία υπολογισμού. Σε αυτή την περίπτωση ο χρόνος επικοινωνίας θα αυξανόταν.

2 Άσκηση

2.1 Το Πρόβλημα

Σε αυτή την άσκηση ζητείται να υλοποιήσουμε ένα πρόγραμμα για τον πολλαπλασιασμό πίνακα επί πίνακα χρησιμοποιώντας το MPI και παραλληλοποίηση του εξωτερικού βρόγχου (ο οποίος διατρέχει τις γραμμές). Κάθε διεργασία αναλαμβάνει τον υπολογισμό μιας «λωρίδας» συνεχόμενων γραμμών του αποτελέσματος (strip-partitioning).

2.2 Μέθοδος Παραλληλοποίησης

Όπως και στην πρώτη άσκηση, έγινε το απαραίτητο include της βιβλιοθήκης του MPI. Στην main ορίσαμε κάποιες μεταβλητές και αμέσως ξεκινήσαμε την περιοχή επικοινωνίας μεταξύ των διεργασιών με την εντολή MPI_Init. Για ακόμη μία φορά ο αριθμός των διεργασιών θα πρέπει να διαιρεί ακριβώς τον αριθμό N των πινάκων.

```
    /* Start MPI */
  ierr = MPI_Init(&argc, &argv);
  if (ierr != 0)
    MPI_Abort(COMMUNICATOR, 1);

  ierr = MPI_Comm_rank(COMMUNICATOR, &my_rank);
  ierr = MPI_Comm_size(COMMUNICATOR, &nprocs);

  if (N % nprocs != 0)
  {
    if (my_rank == BOSS)
      printf("Matrix size not divisible :( \n");
    MPI_Finalize();
    exit(-1);
    /* but thanks for this. */
  }

  work = N / nprocs;
  work_from = my_rank * work;     /* every process offset to start. */
  work_to = (my_rank + 1) * work; /* where should process stop calculating.*/
```

Όπως φαίνεται και στον παραπάνω κώδικα, όλες οι διεργασίες θα έχουν ίδιες σε μέγεθος αλλά και συγκεκριμένες γραμμές για να εργαστούν. Κάθε διεργασία θα ξεκινάει από την γραμμή ανάλογα με την

ταυτότητα τους, δηλαδή η διεργασία 0 -για nprocs=10- θα ξεκινήσει από την γραμμή 0 και θα σταματήσει πριν την 1 * 10 = 10 (δηλαδή 9). Καθώς την γραμμή 10 θα την αναλάβει η διεργασία 1.

Πλέον, σε όλες οι διεργασίες έχει ανατεθεί ένα σύνολο από γραμμές. Μόνο η διεργασία 0 (BOSS) θα διαβάσει τα αρχεία, θα γεμίσει τους πίνακες με τα δεδομένα και θα στείλει σε όλες τις άλλες διεργασίες. Τι χρειάζεται όμως να στείλει;

Εάν παρατηρήσουμε τον αλγόριθμο για τον υπολογισμό μίας τιμής έστω της θέσης (0,0) του πίνακα, παρατηρούμε ότι για τον υπολογισμό της τιμής στην θέση (0,0) χρειάζεται ολόκληρη η γραμμή 0 από τον πίνακα A και ολόκληρη η στήλη 0 από τον B πίνακα. Με αποτέλεσμα η κάθε διεργασία να χρειάζεται όλο τον πίνακα B ενώ μόνο ένα μέρος από τον Πίνακα A.

Για αυτό τον λόγο, η διεργασία 0 στέλνει ολόκληρο τον πίνακα B σε όλες τις άλλες διεργασίες αλλά μόνο ένα μέρος του A. Αυτό το μέρος θα σταλθεί με την εντολή MPI_Scatter όπως φαίνεται παρακάτω.

```
    /* send all with MPI_BCast */
  MPI_Bcast(B, N * N, MPI_INT, BOSS, COMMUNICATOR);
  /* send only the suitable part for every process with MPI_Scatter. */
  MPI_Scatter(A, N * work, MPI_INT, A[work_from], N * work, MPI_INT, BOSS,
    COMMUNICATOR);
```

Όλες οι διεργασίες θα εκτελέσουν ταυτόχρονα τον αλγόριθμο όπως φαίνεται παρακάτω. Κάθε επανάληψη ξεκινά από το offset που έχει υπολογιστεί λίγες γραμμές παραπάνω.

```
    /* Parallel Computation from every process (including BOSS of course). */
  for (i = work_from; i < work_to; i++)
    for (j = 0; j < N; j++)
    {
      for (k = sum = 0; k < N; k++)
        sum += A[i][k] * B[k][j];
      C[i][j] = sum;
    }
```

Τέλος, όλες οι διεργασίες έχουν τελειώσει τον υπολογισμό του δικό τους πίνακα C και πρέπει να στείλουν το αποτέλεσμα στην διεργασία 0. Αυτό επιτυγχάνεται με την εντολή MPI_Gather. Στην ουσία είναι το αντίθετο από το MPI_Scatter που χρησιμοποιήσαμε για την αποστολή. Η διεργασία 0 αφού συλλέξει όλο τον πίνακα C, θα τον μεταφέρει σε ένα αρχείο και το πρόγραμμα μας θα τελειώσει με την εντολή MPI_Finalize().

```
    /* every process will send their C matrix to process BOSS. */
  MPI_Gather(C[work_from], N * work, MPI_INT, C, N * work, MPI_INT, BOSS,
        COMMUNICATOR);
```

2.3 Πειραματικά αποτελέσματα - μετρήσεις

Η χρονομέτρηση πραγματοποιήθηκε με την εντολή MPI_Wtime(). Κάθε πείραμα εκτελέστηκε τέσσερις φορές και υπολογίστηκαν οι μέσοι όροι. Χρονομετρήθηκε η επικοινωνία (Communication) μεταξύ διεργασιών καθώς και η υλοποίηση της συνάρτησης υπολογισμού τελικού πίνακα (Computation). Με την χρήση δύο κόμβων (άρα 8 πυρήνες) έχουμε τα αποτελέσματα:

Number of processes	Communication	Computation	Total
8	0,64	1,17	1,81
	0,74	1,02	1,76
	0,31	1,18	1,49
	1,70	0,90	2,6
summary:	**0,8**	**1,06**	**1,86**

Ενώ με την χρήση τεσσάρων κόμβων (16 πυρήνες) έχουμε:

Number of processes	Communication	Computation	Total
16	1,70	0,43	2,13
	0,74	0,52	1,26
	0,70	0,43	1,13
	0,69	0,59	1,28
summary:	**0,95**	**0,49**	**1,44**

Τέλος με την χρήση οχτώ κόμβων (32 πυρήνες) έχουμε τα εξής:

Number of processes	Communication	Computation	Total
32	3,35	0,14	3,49
	3,25	0,32	3,57
	4,61	0,14	4,75
	2,73	0,30	3,03
summary:	**3,4**	**0,22**	**3,62**

Επίσης ο χρόνος για την εκτέλεση του σειριακού προγράμματος είναι: **4,46** δευτερόλεπτα. Τα αποτελέσματα απεικονίζονται στο παρακάτω γράφημα 2.

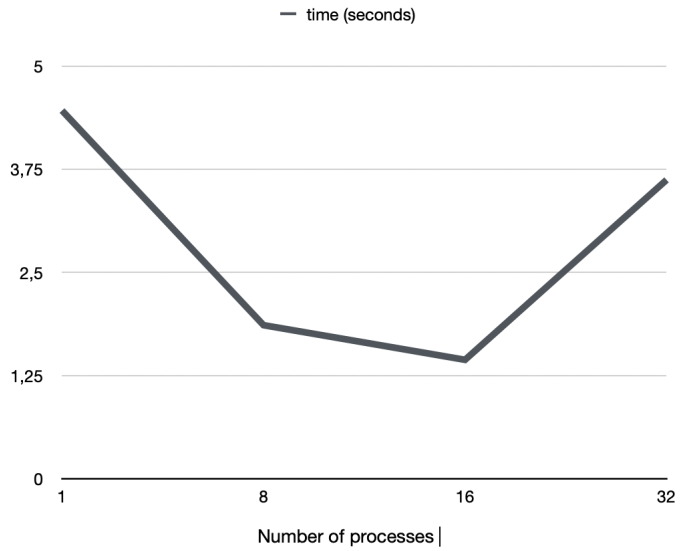

Figure 2: Χρόνος ανά αριθμό διεργασιών

2.4 Σχόλια

Στο παραπάνω γράφημα απεικονίζεται ο χρόνος που απαιτείται για την ολοκλήρωση του προγράμματος ανά εικονική μηχανή. Σημαντική βελτίωση σημειώνεται όταν από μία διεργασία (σειριακό πρόγραμμα) και τα 4,46 δευτερόλεπτα περνάμε στην εικονική μηχανή με δύο κόμβους (άρα 8 πυρήνες/διεργασίες), όπου ο χρόνος μειώνεται κατά 2,6 δευτερόλεπτα. Κάτι που ήταν αναμενόμενο όπως και στην επόμενη εκτέλεση με την δεύτερη εικονική μηχανή και τους 4 κόμβους (άρα 16 πυρήνες/διεργασίες). Παρατηρούμε μία ελάχιστη αύξηση στον χρόνο επικοινωνίας αλλά μία αισθητή μείωση κατά μισό δευτερόλεπτο στον χρόνο υπολογισμού. Άρα ως αποτέλεσμα, η εικονική μηχανή με τους 4 κόμβους να έχει καλύτερη επίδοση.

Στην τρίτη και τελευταία εικονική μηχανή με τους οχτώ κόμβους (32 πυρήνες/διεργασίες) συμπεραίνουμε ότι ο χρόνος υπολογισμού του πίνακα C έχει μειωθεί κατά πολύ. Όμως ο χρόνος της επικοινωνίας, έχει αυξηθεί δραματικά. Ο Μέσος όρος του χρόνου της επικοινωνίας τριπλασιάστηκε και αυτό ήταν αναμενόμενο. Η επικοινωνία μεταξύ πολλών διεργασιών με μεγάλα δεδομένα είναι πιο χρονοβόρο με το να επικοινωνήσουν λιγότερες διεργασίες και ας έχουν μεγαλύτερο φόρτο εργασίας. Άρα ως ιδανική εικονική μηχανή και με τον καλύτερο δυνατό χρόνο εκτέλεσης επιλέγεται η δεύτερη με τους 4 κόμβους με χρόνο 1,44 δευτερόλεπτα.

3 Άσκηση

3.1 Εισαγωγή στις μονόπλευρες επικοινωνίες

Οι μονόπλευρες επικοινωνίες (one-sided communication) που παρέχει το MPI έχουν ως στόχο να παρέχουν την ευκολία της άμεσης προσπέλασης απομακρυσμένων μνημών, καθώς και τη δυνατότητα για υψηλότερες επιδόσεις από αυτές που παρέχουν οι αμφίπλευρες επικοινωνίες (two-sided). Στην μονόπλευρη επικοινωνία, ενεργεί μόνο ένα από τα δύο μέρη είτε ο αποστολέας είτε ο ο παραλήπτης. Σε αυτή την περίπτωση, εκείνος που εκτελεί την επικοινωνία θα πρέπει να προσδιορίσει όλες τις απαραίτητες παραμέτρους. Στη μονόπλευρη αποστολή για παράδειγμα, η πηγή καθορίζει τον χώρο που βρίσκεται το μήνυμα που θα σταλθεί αλλά και τον χώρο του παραλήπτη όπου θα αποθηκευτεί, όμως η αποστολή θα γίνει χωρίς την παρέμβαση ή μεσολάβηση του παραλήπτη. Η δυνατότητα των μονόπλευρων επικοινωνιών παρουσιάστηκε στην έκδοση 2.0 του MPI ενώ στην έκδοση 3.0 προστέθηκαν κάποιες επιπλέον δυνατότητες. Επίσης, είναι ευρέως γνωστές και με τον όρο *απομακρυσμένη προσπέλαση μνήμης* (Remote Memory Access) όπου μία διεργασία μπορεί να επεξεργαστεί, διαβάζοντας ή γράφοντας, άμεσα τη μνήμη μίας άλλης.

3.1.1 Παράθυρα μνήμης και χρονική περίοδος

Στις μονόπλευρες επικοινωνίες, η διεργασία που πραγματοποιεί τη διαδικασία της απομακρυσμένης προσπέλασης των δεδομένων ορίζεται ως προέλευση, ενώ η διεργασία της οποίας η μνήμη προσπελαύνεται ορίζεται ως στόχος. Ως παράθυρο μνήμης ονομάζεται η περιοχή μνήμης της διεργασίας στόχου όπου η διεργασία προέλευσης μπορεί να προσπελάσει. Ο χώρος αυτός καθορίζεται με την κλήση MPI_Win_Create(), η οποία είναι συλλογική κλήση στην ομάδα που ανήκει η διεργασία. Οι διεργασίες προέλευσης όμως, δε μπορούν να προσπελάσουν οποιαδήποτε στιγμή το παράθυρο μνήμης.

Το MPI χρησιμοποιεί το epoch, με το οποίο καθορίζει την περίοδο κατά την οποία αυτή η προσπέλαση είναι εφικτή. Καθ' όλη την διάρκεια που η διεργασία προέλευσης μπορεί να προσπελάσει τα δεδομένα του παραθύρου ονομάζεται περίοδος έκθεσης (exposure epoch). Επιπλέον, η διεργασία προέλευσης θα πρέπει να ορίσει ένα χρονικό διάστημα κατά το οποίο θα προσπελάσει το παράθυρο μνήμης. Αυτό ονομάζεται περίοδος προσπέλασης (access epoch). Μόλις η προσπέλαση ολοκληρωθεί, τερματίζονται οι χρονικοί περίοδοι προσπέλασης προέλευσης και στόχου και έπειτα καμία διεργασία δε μπορεί να προσπελάσει το παράθυρο μνήμης.

3.1.2 Επικοινωνία

Από τη στιγμή που έχουν καθοριστεί τα παράθυρα μνήμης, η επικοινωνία για την μονόπλευρη αποστολή πραγματοποιείται με την συνάρτηση MPI_Put() ενώ για την μονόπλευρη λήψη με την συνάρτηση MPI_Get(). Ενώ για την απομακρυσμένη ενημέρωση των δεδομένων χρησιμοποιείται η συνάρτηση MPI_Accumulate().

Οι παραπάνω επικοινωνίες είναι μη παρεμποδιστικές διαδικασίες (non blocking). Επομένως κατά την κλήση τους, εκκινούν την διαδικασία της προσπέλασης και επιστρέφουν. Επίσης, απαιτούν κατάλληλες κλήσεις συγχρονισμού λόγω περιορισμών.

3.1.3 Συγχρονισμός

Όταν μία κλήση απομακρυσμένης προσπέλασης επιστρέψει, δεν έχουμε εξασφαλίσει ότι η λειτουργία της ολοκληρώθηκε. Για αυτό τον λόγο θα χρειαστούμε κάποιο συγχρονισμό. Οι λειτουργίες συγχρονισμού που παρέχει το MPI εμπλέκονται και οι δύο διεργασίες προέλευσης και στόχου. Με την κλήση της συλλογικής συνάρτησης MPI_Win_fence() από την διεργασία προέλευσης γίνεται η εκκίνηση ή ο τερματισμός της περιόδου προσπέλασης. Ενώ όταν την καλεί η διεργασία στόχου ξεκινάει ή τερματίζει την περίοδο έκθεσης.

Ένας δεύτερος μηχανισμός που ορίζεται από το MPI είναι ο συγχρονισμός μόνο ενός υποσυνόλου των διεργασιών που επικοινωνούν. Αυτή είναι και η βασική διαφορά με τον προηγούμενο μηχανισμό ο οποίος ήταν συλλογικός με αποτέλεσμα άσκοπων επικοινωνιών μεταξύ διεργασιών. Οπότε, μία διεργασία που θέλει να θέσει ένα παράθυρο σε περίοδο έκθεσης καλεί την συνάρτηση MPI_Win_post ενώ μία διεργασία όπου θέλει να προσπελάσει ένα παράθυρο μνήμης καλεί την συνάρτηση MPI_Win_Start. Τέλος, όταν η διεργασία προέλευσης ολοκληρωθεί, καλεί την συνάρτηση MPI_Win_complete και τερματίζει την περίοδο προσπέλασης. Η διεργασία στόχος, που έχει θέσει ένα παράθυρο σε περίοδο έκθεσης καλεί την συνάρτηση MPI_Win_wait και την τερματίζει.

Με τους παραπάνω μηχανισμούς, και οι δύο διεργασίες συμμετείχαν στον συγχρονισμό, καλώντας η κάθε μία την απαραίτητη συνάρτηση. Σε έναν τρίτο μηχανισμό, η επικοινωνία πραγματοποιείται μόνο από την διεργασία προέλευσης. Με την συνάρτηση MPI_Win_lock αποκτά πρόσβαση στο παράθυρο της διεργασίας στόχου. Όταν ολοκληρωθεί, θα χρειαστεί να καλέσει την συνάρτηση MPI_Win_unlock. Μόλις επιστρέψει αυτή το MPI έχει διασφαλίσει ότι όλες οι προσπελάσεις στο παράθυρο έχουν ολοκληρωθεί.

3.2 Μονόπλευρες επικοινωνίες στη χρήση

3.2.1 Διαχείριση παραθύρων

Μία διεργασία για να επιτρέψει μία άλλη να έχει πρόσβαση στην μνήμη της θα πρέπει να δημιουργήσει ένα κατάλληλο παράθυρο. Αυτό επιτυγχάνεται με την κλήση της παρακάτω συνάρτησης.

```
1     int MPI_Win_Create(void *base, MPI_Aint size, int disp_unit,
2         MPI_Info info, MPI_Comm comm, MPI_Win *win);
```

Οι παράμετροι της συνάρτησης είναι: base : ή αρχική διεύθυνση του παραθύρου, size : το μέγεθος του παραθύρου σε bytes, disp_unit : το μέγεθος μιας βασικής μονάδας παραθύρου σε bytes, info : πληροφορίες

για το παράθυρο, comm : ο communicator των διεργασιών, win : δείκτης στο αντικείμενο παραθύρου που επιστρέφεται.
Για ανάκτηση πληροφοριών πρέπει να καλεστεί η συνάρτηση MPI_Win_get_attr με τα ορίσματα:

```
1     int MPI_Win_get_attr(MPI_Win win, int keyVal, void * attrVal,
2             int *flag);
```

ενώ για διαγραφή ενός παραθύρου καλούμε την συνάρτηση MPI_Win_free(MPI_Win *win);

3.2.2 Απομακρυσμένη προσπέλαση δεδομένων

Όπως προαναφέρθηκε παραπάνω οι τρεις μη παρεμποδιστικές συναρτήσεις για την προσπέλαση των δεδομένων είναι οι εξής.

```
1     int MPI_Put(void *origin_addr, int origin_count,
2         MPI_Datatype origin_dtype, int target_rank,
3         MPI_Aint target_disp, int target_count,
4         MPI_Datatype target_dtype, MPI_Win win);
```

Το όρισμα origin_addr είναι ένας δείκτης στην περιοχή της μνήμης όπου είναι τοποθετημένα τα δεδομένα. Η παράμετρος origin_count είναι ο αριθμός στοιχείων που περιέχει η περιοχή της μνήμης με το origin_dtype να είναι ο τύπος των στοιχείων. Το όρισμα target_rank είναι η ταυτότητα της απομακρυσμένης διεργασίας. Ως win περιγράφεται το παράθυρο μνήμης με target_count στοιχεία τύπου target_dtype. Η παράμετρος target_disp έχει τον ρόλο ως offset του παραθύρου. Εάν δηλαδή βάλουμε έναν θετικό αριθμό, μπορούμε να αναφερθούμε σε μία συγκεκριμένη θέση της μνήμης. Με παρόμοια ορίσματα ορίζονται και οι συνάρτηση MPI_Get και MPI_Accumulate που διαφέρει μόνο στο όρισμα op, όπου αποτελεί μία πράξη (όπως MPI_MAX/MIN/...) που θα εφαρμόσει στα παλαιά και τα νέα δεδομένα.

```
1     int MPI_Get(void *origin_addr, int origin_count,
2         MPI_Datatype origin_dtype, int target_rank,
3         MPI_Aint target_disp, int target_count,
4         MPI_Datatype target_dtype, MPI_Win win);
5
6     int MPI_Accumulate(void *origin_addr, int origin_count,
7         MPI_Datatype origin_dtype, int target_rank,
8         MPI_Aint target_disp, int target_count,
9         MPI_Datatype target_dtype,, MPI_Op op, MPI_Win win);
```

3.2.3 Συναρτήσεις συγχρονισμού

Ως πρώτο μηχανισμό συγχρονισμού είδαμε την συλλογική συνάρτηση MPI_Win_fence όπου καλείται από όλες τις διεργασίες που συσχετίζονται με το παράθυρο μνήμης. Επίσης η συγκεκριμένη συνάρτηση πρέπει να καλείται πάντα δύο φορές, μία πριν την έναρξη των λειτουργιών και μία μετά τον τερματισμό.

```
1     int MPI_Win_fence(int assert, MPI_Win win);
```

Ο δεύτερος μηχανισμός που είδαμε χρησιμοποιεί τέσσερις συναρτήσεις.

```
1     int MPI_WIN_post(MPI_Group grp, int assert, MPI_Win win);
2     int MPI_WIN_start(MPI_Group grp, int assert, MPI_Win win);
3     int MPI_Win_complete(MPI_Win win);
4     int MPI_Win_wait(MPI_Win win);
```

Η πρώτη συνάρτηση καλείται από την διεργασία στόχο για να θέσει το παράθυρο σε περίοδο έκθεσης. Η δεύτερη, καλείται από την διεργασία προέλευσης για να δηλώσει την αρχή της περιόδου ενώ η τρίτη συνάρτηση δηλώνει τον τερματισμό. Η τελευταία συνάρτηση καλείται από την διεργασία στόχο ώστε να τερματίσει την περίοδο έκθεσης.

Ως τελευταίο μηχανισμό είδαμε την επικοινωνία που πραγματοποιείται μόνο από την διεργασία προέλευσης χωρίς την συμμετοχή της διεργασίας στόχου. Οι δύο συναρτήσεις είναι :

```
1     int MPI_Win_lock(int lock_type, int rank, int assert, MPI_Win win);
2     int MPI_Win_unlock(int rank, MPI_Win win);
```

MPI_LOCK_SHARED ή MPI_LOCK_EXCLUSIVE είναι οι τιμές που μπορεί να πάρει η μεταβλητή lock_type, όπου ενημερώνει εάν άλλες διεργασίες έχουν πρόσβαση στο παράθυρο μνήμης ταυτόχρονα ή όχι. Η τιμή rank καθορίζει την διεργασία στόχο της οποίας το παράθυρο θα προσπελαστεί. Η παράμετρος assert επιτρέπει τον καθορισμό επιπρόσθετων πληροφοριών για τον συγχρονισμό και η προκαθορισμένη τιμή είναι 0.