Παράλληλα Συστήματα και Προγραμματισμός - ΜΥΕ023
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:

```
   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:

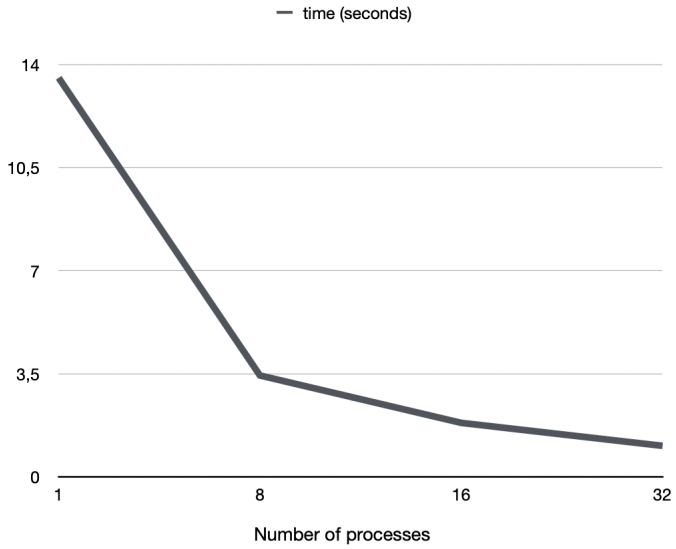

Figure 1: Χρόνος ανά αριθμό διεργασιών

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.

```
1     /* Start MPI */
2   ierr = MPI_Init(&argc, &argv);
3   if (ierr != 0)
4     MPI_Abort(COMMUNICATOR, 1);
5
6   ierr = MPI_Comm_rank(COMMUNICATOR, &my_rank);
7   ierr = MPI_Comm_size(COMMUNICATOR, &nprocs);
8
9   if (N % nprocs != 0)
10  {
11    if (my_rank == BOSS)
12      printf("Matrix size not divisible :( \n");
13    MPI_Finalize();
14    exit(-1);
15    /* but thanks for this. */
16  }
17
18  work = N / nprocs;
19  work_from = my_rank * work;     /* every process offset to start. */
20  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 τελείωσει 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.

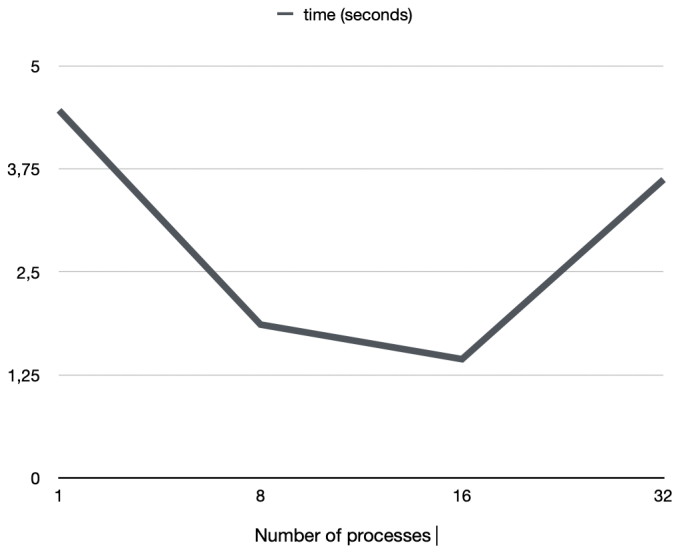

Figure 2: Χρόνος ανά αριθμό διεργασιών

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 (Μονόπλευρες επικοινωνίες)