ΜΥΕ023–Παράλληλα Συστήματα και Προγραμματισμός
Σετ Ασκήσεων #2

Δημήτριος Σίντος
ΑΜ:4012
E-mail: cs04012@uoi.gr

Το 2ο σετ ασκήσεων αφορά στον παράλληλο προγραμματισμό κάνοντας χρήση του MPI. Ζητείται η παραλληλοποίηση 2 εφαρμογών και η αναζητήση πληροφοριών για μονόπλευρες επικοινώνιες.

Όλες οι μετρήσεις έγιναν από το παρακάτω σύστημα:

Όνομα υπολογιστή	opti3060ws10
Επεξεργαστής	Intel i3-8300 3.70Ghz
Πλήθος πυρήνων	4
Μεταφραστής	gcc v7.5.0

Άσκηση 1

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιήθει ο υπολογισμός πρώτων αριθμών. Η βασική ιδέα είναι κάθε διεργασία να αναλαμάνει να ελέγχει ένα υποσύνολο των αριθμών.

Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα από την ιστοσελίδα του μαθήματος και τροποποιήθηκε κατάλληλα. Στην αρχή της main γίνεται διαφοροποίηση των διεργασιών με την MPI_Comm_rank και την MPI_Comm_size. Ο βρόχος που πρέπει να παραλληλοποιηθεί γίνεται "for (i = myid; i < (UPTO-1)/2; i+= nproc)" και έτσι κάθε διεργασία αναλάμβανει να ελέγχει ένα υποσύνολο των αριθμών, μετράει τον αριθμό των πρώτων αριθμών που έχει βρει καθώς και τον μεγαλύτερο από αυτούς. Κάνοντας χρήση των συλλογικών επικοινωνιών και συγκεκριμένα της MPI_Reduce η διεργασία με id 0 συλλέγει τα αποτελέσματα από όλες τις άλλες διεργασίες. Συγκεκριμένα με την πρώτη MPI_Reduce συλλέγουμε από όλες τις διεργάσιες την μεταβλητή count και τις αθροίζουμε στην μεταβλητή counter με την χρήση του ορίσματος MPI_SUM. Στην δεύτερη MPI_Reduce συλλέγουμε από όλες τις διεργάσιες την μεταβλητή lastprime και κρατάμε την μεγαλύτερη στην μεταβλητή lstprm με την χρήση του ορίσματος MPI_MAX. Τέλος η διεργασία με id 0 αναλαμβάνει να τυπώσει στην οθόνη του χρήστη το counter και το lstprm καθώς και τους χρόνους που έκαναν για να υπολογιστούν.

Πειραματικά αποτελέσματα - μετρήσεις

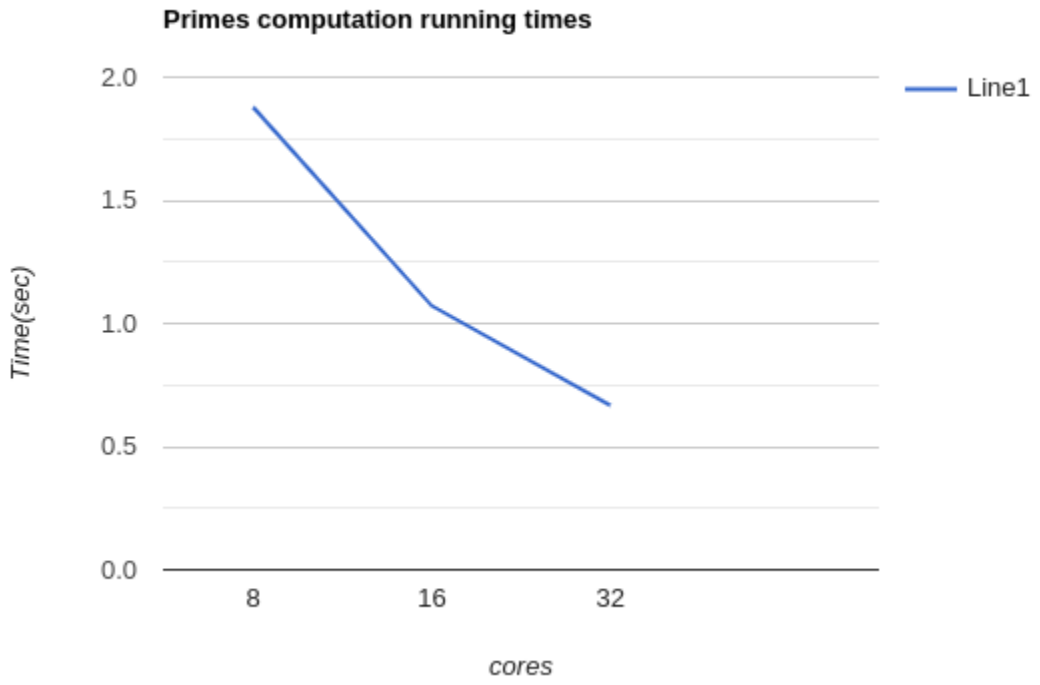
Το πρόγραμμα εκτελέστηκε σε τρεις διαφορετικές εικονικές μηχανές που φτιαχτήκαν μέσω των αρχείων hostfile2, hostfile4 και hostfile8 με 2, 4 και 8 υπολογιστές αντίστοιχα η κάθε μια. Οι υπολογιστές που χρησιμοποιήθηκαν ήταν από τον opti3060ws10 μέχρι τον opti3060ws17, όλοι έχουν τον ίδιο επεξεργαστή με το σύστημα που αναφέραμε στην εισαγωγή. Σε κάθε κόμβο της εικονικής μηχανής γίνεται χρήση όλων των πυρήνων. Π.χ. στην τρίτη εικονική μηχανή συμμετέχουν 8 κόμβοι, κάθε ένας εκ των οποίων είναι τετραπύρηνος, το πλήθος των διεργασιών που θα πρέπει να δημιουργήται είναι 8 x 4 = 32.

Η χρονομέτρηση έγινε με τη συνάρτηση MPI_Wtime().Μετρήθηκε ο συνολικός χρόνος εκτέλεσης καθώς και ο χρόνος των overheads αφαιρώντας από τον συνολικό (αθροιστικό) χρόνο εκτέλεσης, τους χρόνους υπολογισμών. Κάθε πείραμα εκτελέστηκε 4 φορές και υπολογίστηκαν οι μέσοι όροι. Τα αποτελέσματα δίνονται στον παρακάτω πίνακα (οι χρόνοι είναι σε sec):

Πυρήνες - διεργασίες	**1η εκτέλεση**		**2η εκτέλεση**		**3η εκτέλση**		**4η εκτέλεση**		**Μέσος όρος**	
	Total	**overheads**	**Total**	**overheads**	**Total**	**overheads**	**Total**	**overheads**	**Total**	**overheads**
8	1.881699	0.167478	1.866257	0.153355	1.873810	0.156959	1.890517	0.186869	**1.87807075**	**0.16616525**
16	1.101896	0.227934	1.062587	0.211066	1.071298	0.172275	1.055360	0.167039	**1.07278525**	**0.1945785**
32	0.710365	0.267771	0.649672	0.221446	0.647849	0.197648	0.659342	0.230608	**0.666807**	**0.22936825**

Χρόνος σειριακού προγράμματος : **13.331740**

Με βάση τους πίνακες των αποτελεσμάτων προκύπτει η παρακάτω γραφική παράσταση.



Σχόλια

Με βάση τα αποτελέσματα βλέπουμε ότι η αύξηση του αριθμού των πυρήνων μειώνει σημαντικά τους χρόνους εκτέλεσης και μάλιστα περίπου ιδανικά, διότι άμα παρατηρήσουμε τον χρόνο που έκαναν οι 16 διεργασίες είναι περίπου ο μισός από αυτόν που έκαναν οι 8 (το ίδιο ισχύει αντίστοιχα και με τις 32 σε σχέση με τις 16). Όλα αυτά ήταν αναμενόμενα διότι οι επαναλήψεις μοιράστηκαν ισόποσα και κάθε μία είχε παρόμοιο φόρτο υπολογισμού με τις άλλες.

Ο χρόνος που σπαταλούν οι διεργασίες στους υπολογισμούς είναι και ο κύριος χρόνος από τον συνολικό αφού ο χρόνος για τα overheads είναι πολύ μικρότερος αλλά όχι αμελητέος.

Επίσης θα υπάρχει και κάποιοι “άχρηστοι” χρόνοι λόγω ανισοκατανομής φόρτου, που δεν εμφανίζονται στη σειριακή εκτέλεση. Αυτοί οι χρόνοι δεν συμπεριλαμβάνονται στα overheads αλλά όσο αυξάνονται οι διεργασίες αυτοί οι χρόνοι λόγω ανισοκατανομής μειώνονται.

Άσκηση 2

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιήθει ένα πρόγραμμα για τον πολλαπλασιασμό πίνακα επί πίνακα. Η βασική ιδέα είναι η παραλληλοποίηση του εξωτερικού βρόχου (ο οποίος διατρέχει τις γραμμές). Η παραλληλοποίηση αυτή είναι γνωστή και ως μέθοδος διαχωρισμού γραμμών (strip-partitioning) όπου κάθε διεργασία αναλαμβάνει τον υπολογισμό μια «λωρίδας» συνεχόμενων γραμμών του αποτελέσματος.

Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα από την ιστοσελίδα του μαθήματος και τροποποιήθηκε κατάλληλα. Στην αρχή της main γίνεται διαφοροποίηση των διεργασιών με την MPI_Comm_rank και την MPI_Comm_size. Έπειτα η διεργασία με id 0 είναι αυτή που είναι υπεύθυνη για να διαβάσει τους πίνακες A και B, χωρίζει τον A σε ισομερή κομμάτια και αφού έχει γίνει η κατάλληλη αρχικοποίηση του χώρου τον διασκορπίζει, με την συνάρτηση MPI_Scatter(), στις άλλες διεργασίες. Τον B τον εκπέμπουμε προς όλες τις άλλες διεργασίες με την συνάρτηση MPI_Bcast(). Εδώ πέρα θα μπορούσε να γίνει κάποια συζήτηση: Μια εναλλακτική λύση θα ήταν να όλες οι διεργασίες να διαβάζουν τον B και όχι μόνο η διεργασία με id 0. Με αυτόν τον τρόπο θα κερδίζαμε σίγουρα χρόνο αφού στην εκφώνηση του σετ αναφέρει να μην μετρηθούν οι χρόνοι ανάγνωσης αρχείων. Επίσης δεν θα χρειαζόταν το MPI_Bcast(), οπότε οι επικοινωνίες θα μειώνονταν. Παρόλα αυτά επιλέξαμε την πρώτη περίπτωση ώστε παρακάτω στην χρονομέτρηση του προγράμματος να γίνει πιο φανερό ότι οι επικοινωνίες μεταξύ των διεργασιών είναι εχθρός της ταχύτητας!

Στην συνέχεια του προγράμματος κάθε διεργασία κάνει τους δικούς υπολογισμούς και αποθηκεύει το αποτέλεσμα στον πίνακα Cc που είχε αρχικοποιηθεί παραπάνω. Με την συνάρτηση MPI_Gather() η διεργασία με id 0 συλλέγει στον πίνακα C όλα τα κομμάτια του πίνακα που έχουν υπολογιστεί. Τέλος η διεργασία με id 0 αναλαμβάνει να γράψει τα αποτελέσματα σε αρχείο καθώς και τους χρόνους που έκαναν για να υπολογιστούν.

Πειραματικά αποτελέσματα - μετρήσεις

Για να ελέγξουμε την σωστή λειτουργία του παράλληλου προγράμματος δημιουργήσαμε ένα απλό πρόγραμμα που συγκρίνει τον σωστό πίνακα του σειριακού προγράμματος με αυτόν που επέστρεφε το παράλληλο. Αφού βεβαιωθήκαμε ότι δουλεύει σωστά εκτελέσαμε το πρόγραμμα σε τρεις διαφορετικές εικονικές μηχανές ίδιες με αυτές της **Άσκησης 1.**

Η χρονομέτρηση έγινε με τη συνάρτηση MPI_Wtime().Μετρήθηκε ο συνολικός χρόνος εκτέλεσης καθώς και ο χρόνος των overheads αφαιρώντας από τον συνολικό (αθροιστικό) χρόνο εκτέλεσης, τους χρόνους υπολογισμών. Κάθε

πείραμα εκτελέστηκε 4 φορές και υπολογίστηκαν οι μέσοι όροι. Τα αποτελέσματα δίνονται στους παρακάτω πίνακες (οι χρόνοι είναι σε sec):

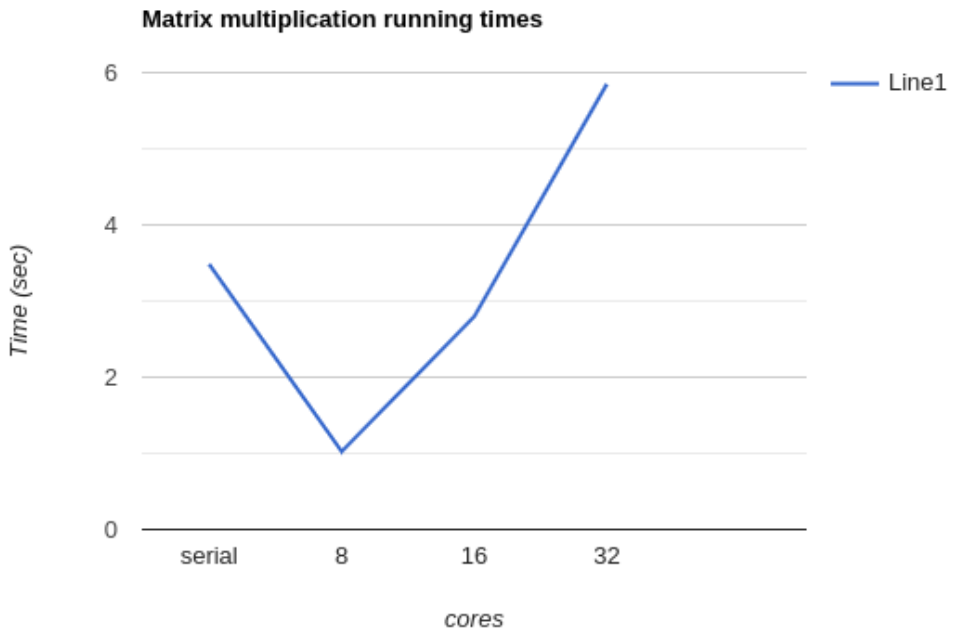
Πυρήνες - διεργασίες	**1η εκτέλεση**		**2η εκτέλεση**		**3η εκτέλση**		**4η εκτέλεση**		**Μέσος όρος**	
	Total	**overheads**	**Total**	**overheads**	**Total**	**overheads**	**Total**	**overheads**	**Total**	**overheads**
8	1.012171	0.488538	0.928672	0.401597	1.193617	0.672760	0.962177	0.430048	**1.02415925**	**0.49823575**
16	1.770389	1.505706	3.229963	2.961685	2.940077	2.678532	3.236349	2.975384	**2.7941945**	**2.53032675**
32	5.542648	5.408830	5.501028	5.366579	7.683134	7.550900	4.670893	4.540966	**5.84942575**	**5.71681875**

Χρόνοι υπολογισμών:

Πυρήνες - διεργασίες	**1η εκτέλεση** **Computations time**	**2η εκτέλεση** **Computations time**	**3η εκτέλση** **Computations time**	**4η εκτέλεση** **Computations time**	**Μέσος όρος** **Computations time**
8	0.523633	0.527076	0.520857	0.532129	**0.52592375**
16	0.264682	0.268278	0.261545	0.260966	**0.26386775**
32	0.133817	0.134449	0.132233	0.129927	**0.1326065**

Χρόνος σειριακού προγράμματος : **3.486525**

Με βάση τους πίνακες των αποτελεσμάτων προκύπτει η παρακάτω γραφική παράσταση.



Σχόλια

Με βάση τα αποτελέσματα βλέπουμε ότι τους καλύτερους χρόνους τους επιτυγχάνουμε στο παράλληλο πρόγραμμα που τρέχει στην εικονική μηχανή με 8 διεργασίες. Όσο αυξάνεται ο αριθμός των πυρήνων ο χρόνος υπολογισμών μειώνεται όπως περιμένουμε, δηλαδή στο μισό από ότι στην προηγούμενη περίπτωση. Όμως αυτό δεν συμβαίνει για τον συνολικό χρόνο αφού μετά τις 8 διεργασίες ο χρόνος αυξάνεται διαρκώς μέχρι οπού είναι και χειρότερος από τον σειριακό. Αυτό είναι αναμενόμενο, αφού βλέποντας τους χρόνους των overheads βλέπουμε ότι διαρκώς αυξάνονται σε κάθε περίπτωση. Ο λόγος είναι οι επικοινωνίες μεταξύ των διεργασιών. Όσο περισσότερες είναι οι διεργασίες τόσο αυξάνεται ο χρόνος που απαιτούν για να επικοινωνούν μεταξύ τους.

Άσκηση 3

Μονόπλευρες επικοινωνίες

Η επικοινωνία μεταξύ δύο διεργασιών, όπως την έχουμε δει μέχρι στιγμής, θεωρείται αμφίπλευρη (two-sided) διότι, για να ολοκληρωθεί, πρέπει να ενεργήσουν και οι δύο εμπλεκόμενες διεργασίες· η μία διεργασία πρέπει να εκτελέσει αποστολή και η άλλη λήψη. Στη λεγόμενη μονόπλευρη επικοινωνία (one-sided coommunication), ενεργεί μόνο ένα από τα δύο μέρη είτε ο αποστολέας είτε ο παραλήπτης. Στην περίπτωση αυτή, εκείνος που εκτελεί την επικοινωνία πρέπει να προσδιορίσει όλες τις απαραίτητες παραμέτρους. Για παράδειγμα, στη μονόπλευρη αποστολή, η πηγή πρέπει να καθορίσει τον χώρο που βρίσκεται το μήνυμα που θα σταλθεί αλλά και τον χώρο (στον προορισμό) όπου θα αποθηκευτεί και η αποστολή θα γίνει χωρίς τη μεσολάβηση του παραλήπτη. Οι μονόπλευρες επικοινωνίες είναι

ευρύτερα γνωστές με τον όρο απομακρυσμένη προσπέλαση μνήμης (Remote Memory Access, rma), όπου μία διεργασία μπορεί να προσβεί, γράφοντας ή διαβάζοντας, άμεσα τη μνήμη μίας άλλης.

Το mpi πρώτη φορά παρείχε τη δυνατότητα μονόπλευρων επικοινωνιών στην έκδοση 2.0, ενώ η έκδοση 3.0 πρόσθεσε μερικές επιπλέον δυνατότητες.

Προκειμένου μία διεργασία να χρησιμοποιήσει μονόπλευρες επικοινωνίες, πρέπει να κάνει τέσσερα βήματα:

1.Να ορίσει έναν χώρο στη μνήμη της ο οποίος θα είναι διαθέσιμος για απομακρυσμένη πρόσβαση από όλες τις άλλες.Ο χώρος αυτός ονομάζεται παράθυρο μνήμης και καθορίζεται με την κλήση MPI_Win_create(), η οποία είναι συλλογική κλήση στην ομάδα που ανήκει η διεργασία. Το mpi-3 επιπλέον διαθέτει την κλήση MPI_Win_create_dynamic(), η οποία δεν είναι συλλογική και επιτρέπει σε μία διεργασία δυναμικά να καταχωρεί έναν δικό της χώρο ως προσβάσιμο από τις υπόλοιπες.

2. Να γίνει επιτρεπτή η πρόσβαση σε αυτή την μνήμη.

3.Να ξεκινήσει η μονόπλευρη επικοινωνία.

4.Να ολοκληρωθούν όλες οι εκκρεμής λειτουργίες της μονόπλευρης επικοινωνίας.

Παράδειγμα για την MPI_Win_create():

```
MPI_Win win;
double ulocal[MAX_NX][NY+2];

MPI_Win_create( ulocal, (NY+2)*(i_end-i_start+3)*sizeof(double),
           sizeof(double), MPI_INFO_NULL, MPI_COMM_WORLD, &win );
```

Τα ορίσματα της MPI_Win_create είναι με την σειρά: ο πίνακας που δηλώνει τον χώρο της μνήμης, το μέγεθος του πίνακα σε byte, το μέγεθος της μονάδας του πίνακα(στην περίπτωση μας sizeof(double), ένα αντικείμενο “πρόταση” και η επικοινωνία που καθορίζει ποιες είναι οι πιθανές διαδικασίες.

Οι λειτουργίες RMA πραγματοποιούνται μεταξύ δύο “φρουρών”. Ο ένας ξεκινά μια περίοδο όπου επιτρέπεται η πρόσβαση σε ένα αντικείμενο του παραθύρου μνήμης, και ο άλλος την τερματίζει . Αυτές οι περίοδοι ονομάζονται εποχές. Η ευκολότερη συνάρτηση που χρησιμοποιείται για να ξεκινήσει και να τερματίσει τις εποχές είναι η MPI_Win_fence. Η συνάρτηση ονομάζεται "φράχτης" επειδή όλες ο λειτουργίες RMA πριν την πρέπει να ολοκληρωθούν για να ξεκινήσουν αυτές που είναι μετά την fence. Αυτό μπορεί να φαίνεται περίπλοκο, αλλά είναι εύκολο. Στην πράξη, η MPI_Win_fence χρησιμοποιείται για να ξεχωρίζει τις λειτουργίες σε ομάδες.

Τρεις λειτουργίες είναι διαθέσιμες για την έναρξη της μεταφοράς δεδομένων στην μονόπλευρη επικοινωνία. Αυτές είναι οι MPI_Put, MPI_Get και η MPI_Acumulate. Όλες είναι non-blocking. Ολοκληρώνονται στο τέλος της εποχής στην οποία ξεκίνησαν, για παράδειγμα, στο τέλος της MPI_Win_fence. Επειδή αυτές οι ρουτίνες καθορίζουν τόσο την πηγή όσο και τον προορισμό των δεδομένων, έχουν περισσότερα ορίσματα από ότι οι ρουτίνες αμφίπλευρης επικοινωνίας.

Τα επιχειρήματα μπορούν εύκολα να γίνουν κατανοητά λαμβάνοντας τα λίγα κάθε φορά:

1. Τα τρία πρώτα ορίσματα περιγράφουν τα δεδομένα προέλευσης, δηλαδή τα δεδομένα της καλούσας συνάρτησης. Αυτά είναι τα συνήθως ορίσματα όπως "buffer, count, datatype".

2. Το επόμενο όρισμα είναι το id της διεργασίας που θα λάβει το μήνυμα.

3. Τα επόμενα τρία ορίσματα περιγράφουν το buffer προορισμού.

4. Το τελευταίο όρισμα είναι το αντικείμενο παραθύρου μνήμης.

Παράδειγμα ανταλλαγής μηνυμάτων, με μονόπλευρη επικοινωνία:

```
void exchang_nbrs( double u_local[][NY+2], int i_start, int i_end,
                   int left, int right, MPI_Win win )
{
    MPI_Aint left_ghost_disp, right_ghost_disp;
    int      c;

    MPI_Win_fence( 0, win );
    /* Put the left edge into the left neighbors rightmost
       ghost cells.  See text about right_ghost_disp */
    right_ghost_disp = 1 + (NY+2) * (i_end-i-start+2);
    MPI_Put( &u_local[1][1], NY, MPI_DOUBLE,
            left, right_ghost_disp, NY, MPI_DOUBLE, win );
    /* Put the right edge into the right neighbors leftmost ghost
       cells */
    left_ghost_disp = 1;
    c = i_end - i_start + 1;
    MPI_Put( &u_local[c][1], NY, MPI_DOUBLE,
             right, left_ghost_disp, NY, MPI_DOUBLE, win );

    MPI_Win_fence( 0, win )
}
```

Πηγές:

https://repository.kallipos.gr/bitstream/11419/3214/3/9717_chapter05.pdf

http://etutorials.org/Linux+systems/cluster+computing+with+linux/Part+II+Parallel+Programming/Chapter+9+Advanced+Topics+in+MPI+Programming/9.7+Remote+Memory+Access/