Παράλληλοι Αλγόριθμοι και Υπολογιστική Πολυπλοκότητα

4η άσκηση για το σπίτι

Βασιλακοπουλος Θεοδωρος 57826

Ερώτημα 1.

Στο OpenMP είδαμε τη χρήση της ατομικής πράξης (atomic), της κρίσιμης περιοχής (critical) και των κλειδαριών. Ποια είναι κατά τη γνώμη σας η αναγκαιότητα των μηχανισμών αυτών, υπάρχουν κοινά χαρακτηριστικά μεταξύ τους; Πιστεύετε ότι οι μηχανισμοί αυτοί θα είναι αναγκαίοι και στο MPI;

Οπως στο openmp ετσι και σε καθε παραλληλη διαδικασια υπαρχει αναγκη για υπαρξη καποιου ελεγχου race conditions δηλαδη συναγωνισμου νηματων . Στο MPI οι αντιστοιχες συναρτησεις υλοποιούνται ως εξης

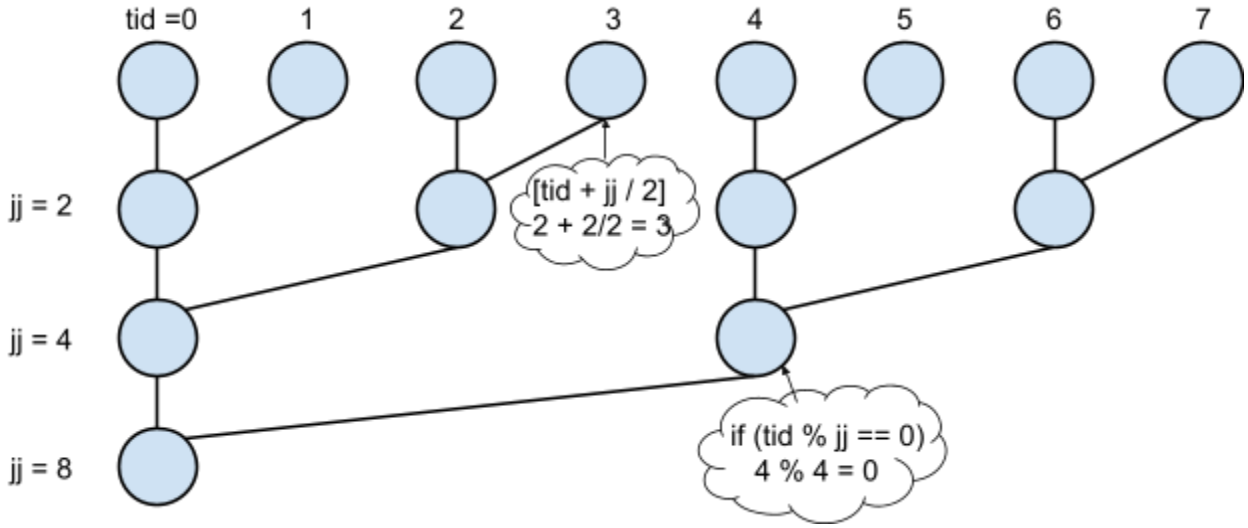
- **MPI_WIN_LOCK**.
- **MPI_Accumulate** performs an atomic update on window data.
- **MPI_Get_accumulate** fetches the value and performs an update.
- **MPI_Fetch_and_op** is similar to **MPI_Get_accumulate** but is a shorthand function for the common case of a single element.
- **MPI_Compare_and_swap** does what the name suggests.

Ερώτημα 2.

Θεωρήστε ένα δισδιάστατο πίνακα Α με MxN ο οποίος περιέχει ακέραιες τιμές. Για να βρουμε το ελαχιστο και το μεγιστο καθε πινακα απλα σε καθε επαναληψη που εχει αναλαβει καθε νημα υπαρχουν δυο ελεγχοι για να αντικατασταθει η προηγουμενη τιμη max ή min. Οι επαναληψεις γινοται στατικα με M/number of threads .

```
#pragma omp parallel shared(nthreads,localData)  private(i,j,ii,jj,tid,temp ) num_threads(nthreads)
        {
                tid = omp_get_thread_num();

                if (tid == 0) {      printf("\n nthreads = %d \n", omp_get_num_threads()); }


                for (i = 0; i < M/nthreads; i++)
                {
                        for (j = 0; j < N; j++)
                        {
                                temp = A[tid * M / nthreads + i][j];

                                if previoious_min is greater than temp -> change ;

                                if previoious_max is smaller than temp -> change ;

                                Add the temp in sum ;


                        }

                }
```

Μετα την ολοκληρωση αυτου του βηματος θα χρειαστει να κανουμε reduction tree ωστε ολα τα αποτελεσματα να ενωθουν σε ενα .




```
//Reduction
#pragma omp barrier
        for (jj = 2; jj < nthreads+1 ; jj *= 2) {
                if (tid % jj == 0) {

                        temp = A[tid + jj / 2];

                        if previoious_min is greater than temp -> change ;;

                        if previoious_max is smaller than temp -> change ;;

                        Add the temp in sum ;

                }
#pragma omp barrier
        }
```