Software & Programming of HPC Systems

ΕΡΓΑΣΙΑ 1: MPI and OpenMP

Φλουρής Παντελής, up1093507

Κολτσάκης Χρυσάφης, up1084671

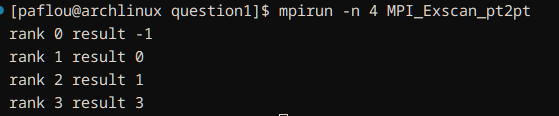
1. **Hybrid Programming Model and Parallel I/O**
2. MPI\_Exscan\_pt2pt.c

Για την υλοποίηση της λειτουργίας MPI\_Exscan χρησιμοποιώντας επικοινωνίες point-to-point (pt2pt), δημιουργήσαμε μια συνάρτηση (MPI\_Exscan\_pt2pt), η οποία δέχεται ως ορίσματα τον συνολικό αριθμό διεργασιών, τον αριθμό της διεργασίας που την καλεί και την τιμή που θα συμμετάσχει στην άθροιση. Αρχικοποιούμε μια μεταβλητή για την τιμή που θα υπολογιστεί σε κάθε διεργασία και μέσω μιας δομής if-else, καθορίζουμε τις ενέργειες:

* Η πρώτη διεργασία στέλνει την τιμή της στην επόμενη διεργασία μέσω της συνάρτησης MPI\_Send, με tag ίσο με τον αριθμό της διεργασίας.
* Οι ενδιάμεσες διεργασίες δέχονται αρχικά την τιμή από την προηγούμενη διεργασία, προσθέτουν τη δική τους τιμή σε αυτήν, και στη συνέχεια στέλνουν το νέο αποτέλεσμα στην επόμενη διεργασία.
* Η τελευταία διεργασία δέχεται απλώς την τιμή από την προηγούμενη διεργασία χωρίς να στείλει κάτι πίσω.

Αυτή η προσέγγιση επιτρέπει την αθροιστική επεξεργασία των τιμών σε κάθε διεργασία, ακολουθώντας την αρχή της exscan.

Παράδειγμα εκτέλεσης



1. MPI\_Exscan\_omp.c

Για αυτό το υποερώτημα αρχικά μετονομάσαμε την συνάρτηση του προηγόυμενου ερωτήματος σε MPI\_Exscan\_omp.

Εφόσον χρειαζόμαστε OpenMP Threads, αντικαθιστούμε το MPI\_Init με MPI\_Init\_threads με MPI\_THREAD\_MULTIPLE, επιτρέποντας σε πολλαπλά threads να εκτελούν MPI εντολές χωρίς περιορισμούς.

*Αν και δεν παρατηρησαμε διαφορά στην ορθή εκτέλεση του προγράμματος αναμεσα στα διαφορετικά επίπεδα υποστήριξης νημάτων (MPI\_THREAD\_SINGLE, MPI\_THREAD\_FUNNELED, MPI\_THREAD\_SERIALIZED, MPI\_THREAD\_MULTIPLE), θεωρούμε πως το MPI\_THREAD\_MULTIPLE ειναι πιο ορθό στην προκειμένη περίπτωση.*

Η νέα συνάρτηση καλείται μεσα απο μια παράλληλη περιοχή στην main, οπου αρχικοποιείται και ο αριθμος του κάθε νήματος. Η παραλληλη περιοχη εχει κοινή μεταβλητή την prev.

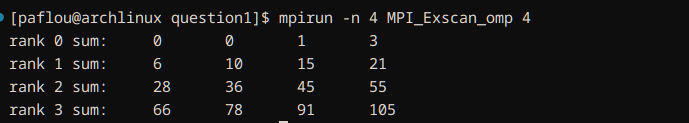
Τα νηματα καθε διεργασίας καλουν μια νεα συναρτηση (for\_loop), η οποία εκτελείται σειριακά μεσω του pragma omp ordered, (πραγμα απαραίτητο καθώς το νήμα x για τον υπολογισμό του απαιτεί να έχει ολοκληρωθει το νημα x-1).

Στην συναρτηση for\_loop, η κάθε διεργασία αποθηκευει στην μεταβλητη sum την τελική τιμή για το νήμα, ενημερώνει το prev για το επόμενο νήμα, και επιστρεφει το sum.

Στην MPI\_Exscan\_omp, αποστέλλεται η κοινή για τα νήματα μεταβλητή prev, απο το τελευταίο νήμα της κάθε διεργασίας στο πρώτο της επόμενης.

Η τελική τιμή του κάθε νήματος επιστρέφεται και αποθηκεύεται σε συγκεκριμένες θέσεις ενός πίνακα στην διεργασία, απ’οπου τυπώνονται στην κονσόλα.

Παράδειγμα εκτέλεσης



1. MPI\_Exscan\_omp\_io.c

Για αυτό το υποερώτημα, οι συναρτήσεις for\_loop και MPI\_Exscan\_omp παρέμειναν ιδιες, και δημιουργήσαμε 2 καινουργιες συναρτησεις:

**initalizeMatrix** -> Αρχικοποιεί το μητρώο με τυχαίες τιμές μεσω rand\_r.

**checkMatrix** -> Διαβάζει Ν\*N\*N θεσεις του δυαδικου αρχείου και ελεγχει εαν ειναι σωστο (χρησιμοποιώντας rand\_r με ιδιο seed με την αρχικοποίηση).

Στην παράλληλη περιοχή της main, γινεται δυναμική δεσμευση μνήμης του μητρώου του κάθε νήματος, η αρχικοποίηση του seed και μετά του μητρώου.

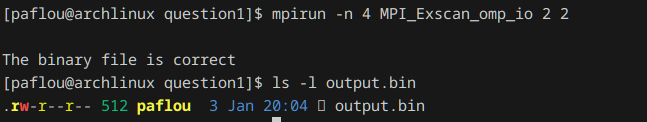
Με την κλήση της MPI\_Exscan\_omp υπολογίζονται τα offsets για το κάθε νήμα, και μεσω της **MPI\_File\_write\_at\_all**, ξεκινούν ολα τα νήματα την εγγραφή ταυτόχρονα.

Στην συνέχεια το κάθε νήμα καλεί την **checkMatrix**, ελέγχοντας αν έγινε σωστά η εγγραφή για το συγκερκιμένο νήμα.

Επειτα τυπώνεται κατάλληλο μήνυμα (επιτυχούς ή μή εγγραφής) και τερματίζει το πρήγραμμα.

Χρησιμοποιούνται **MPI\_File\_write\_at\_all** και **MPI\_File\_read\_at\_all**, πετυχαίνοντας ταυτόχρονη εγγραφή / ανάγνωση του αρχείου απο όλα τα νήματα.

Παράδειγμα εκτέλεσης



1. MPI\_Exscan\_omp\_io\_compressed.c

Οι αλλαγές απο το προηγούμενο υποερώτημα είναι μικρές αλλά ουσιώδεις. Χρησιμοποιούμε την βιβλιοθηκη **zlib** για συμπίεση των δεδομένων με τον εξής τρόπο:

Στην παράλληλη περιοχή, δημιουργείται ενας πίνακας char με μέγεθος BUFFER (στον οποίο θα μπούν τα συμπιεσμένα δεδομένα πριν την εγγραφή), και αρχικοποιείται και μια μεταβλητή που θα αποθηκευτεί το μέγεθος των συμπιεσμένων δεδομένων.

Τα νήματα καλούν ταυτόχρονα την συνάρτηση compress, συμπιέζοντας τα δεδομένα τους, και μετά γράφουν τα συμπιεσμένα δεδομένα στο δυαδικό αρχείο.

Χρειάστηκε να τροποποιήσουμε και την συνάρτηση checkMatrix, ετσι ωστε να αποσυμπιέζει τα δεδομένα προκειμένου να ελέγξει την ορθότητα τους.

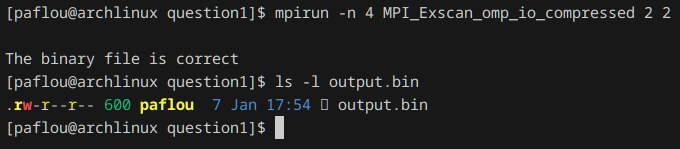
Αρχικοποιούνται μεταβλητές για την αποθήκευση των δεδομένων του συμπιεσμένου αρχείου και για την αποθήκεση των αποσυμπιεσμένων αρχείων, και στην συνέχεια το αρχείο διαβάζεται ταυτόχρονα απο όλα τα νήματα μεσω **MPI\_File\_read\_at\_all** και αποθηκεύεται σε καταλληλη μεταβλητη.

Η μεταβλητή αποσυμπιέζεται με uncompress, και ακολουθεί ο ιδιος κώδικας με το προηγούμενο ερώτημα.

Παρατηρούμε οτι δεν υπάρχει μεγάλη διαφορά στο μέγεθος του δυαδικού αρχείου με την συμπίεση. Για αυτο ευθύνονται τα τυχαία δεδομένα μας, που δεν εχουν πολλά μοτίβα που μπορεί να εκμεταλλευτεί η βιβλιοθήκη.

**Σημείωση: Σε αυτό το ερώτημα, χρήση της MPI\_File\_write\_at\_all είχε σαν αποτέλεσμα deadlock. Υποθέτουμε οτι έχει να κάνει με τα διαφορετικά μεγέθη εγγραφής, καθώς στο προηγούμενο υποερώτημα λειτουργεί. Η ταυτόχρονη εγγραφή υλοποιήθηκε με #pragma omp single και MPI\_Barrier.**

Παράδειγμα εκτέλεσης



Στο συγκεκριμένο παράδειγμα το αρχείο ειναι μεγαλυτερο(!) απο το μη συμπιεσμένο, λογω της ελλειπούς δυνατότητας συμπίεσης.

1. **Parallel Parametric Search in Machine Learning**
2. **OpenMP Tasking**

**Α)** Οι επαναλήψεις του βρόγχου διαμοιράζονται δυναμικά στα νήματα σε ομάδες των 2.

Εστω οτι εχω 4 νήματα. Αρχικά το κάθε νήμα θα πάρει απο 2 συνεχόμενες επαναλήψεις *(π.χ. Ν0 -> 0,1 | Ν1 -> 2,3 | Ν2 -> 4,5 | Ν3->6,7)* και όποτε κάποιο νήμα ολοκληρώνει τις επαναλήψεις του του ανατείθενται οι 2 επόμενες δυναμικά *(εαν το Ν2 ολοκληρώσει πρώτο τις επαναλήψεις του 4,5 του ανατείθενται οι επαναλήψεις 8,9 κ.ο.κ)*. Συνεχίζεται έτσι μέχρι την ολοκλήρωση του βρόγχου.

**Β)** Η ισοδύναμη υλοποίηση βρισκεται στο αρχείο **tasking.c**.