Software & Programming of HPC Systems

ΕΡΓΑΣΙΑ 1: MPI and OpenMP

Φλουρής Παντελής, up1093507

Κολτσάκης Χρυσάφης, up1084671

1. Hybrid Programming Model and Parallel I/O

1) 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, την τιμή που θα συμμετάσχει στην άθροιση και το operation. Αρχικοποιούμε μια μεταβλητή για την τιμή που θα υπολογιστεί σε κάθε διεργασία και σε εναν βρόγχο επανάληψης οι διεργασίες υπολογίζουν τις τιμές τους μέσω **Recursive Doubling,** αποφεύγοντας την σειριακή επικοινωνία των διεργασιών και επιτυγχάνοντας O(logN) πολυπλοκότητα.

Παράδειγμα εκτέλεσης με τιμή κάθε rank = rank και MPI_Op = MPI_SUM

```
-> % mpirun -n 7 MPI_Exscan_pt2pt
rank 0 result -1
rank 1 result 0
rank 2 result 1
rank 3 result 3
rank 4 result 6
rank 5 result 10
rank 6 result 15
```

2) 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_SINGLE, καθώς θα χρησιμοποιήσουμε μόνο το πρώτο νήμα της κάθε διεργασίας για την MPI επικοινωνία.

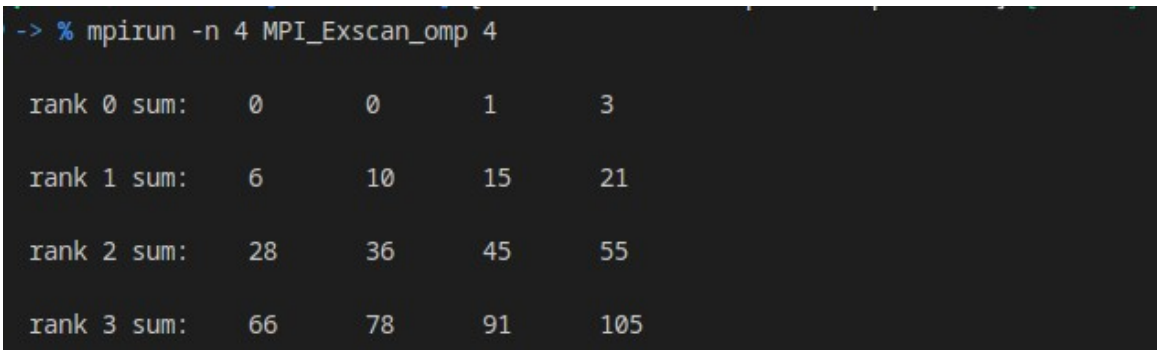
Τα νήματα της κάθε διεργασίας αποθηκεύουν την τιμή τους σε θέσεις ενός δισδιάστατου (για αποφυγή false sharing) πίνακα, που ειναι κοινός ανάμεσα στα νήματα της διεργασίας.

Η νέα συνάρτηση καλείται μεσα απο μια παράλληλη περιοχή στην main, οπου αρχικοποιείται και ο αριθμος του κάθε νήματος. Η παραλληλη περιοχη εχει κοινούς τους πίνακες sum και values, που περιέχουν το τελικό αποτέλεσμα και την τιμή του κάθε νήματος αντίστοιχα.

Στην συνάρτηση αρχικά υπολογίζονται μέσω ενός for loop τα τοπικά αθροίσματα / γινόμενα / μέγιστα / ελάχιστα του κάθε νήματος της διεργασίας. Στην συνέχεια μέσω Recursive Doubling, αποστέλλονται από ένα νήμα της κάθε διεργασίας η τιμή του τελευταίου νήματος συν / επί / το μεγιστο/ελαχιστο το άθροισμα του τελευταίου νήματος της διεργασίας.

Με αντίστοιχο τρόπο η κάθε διεργασία παραλαμβάνει τα δεδομένα και ενημερώνει τον πίνακα sum (έτσι ώστε όλα τα νήματα να έχουν την σωστή τιμή).

Παράδειγμα εκτέλεσης



3) MPI_Exscan_omp_io.c

Για αυτό το υποερώτημα, από την συνάρτηση MPI_Exscan_omp κρατήσαμε μόνο το ενδεχόμενο MPI_SUM, καθώς θέλουμε να υπολογίσουμε τα offsets για κάθε νήμα. Δημιουργήσαμε 2 καινούργιες συναρτήσεις:

initalizeMatrix -> Αρχικοποιεί το μητρώο με τυχαίες τιμές μεσω rand_r.

checkMatrix -> Διαβάζει N*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.

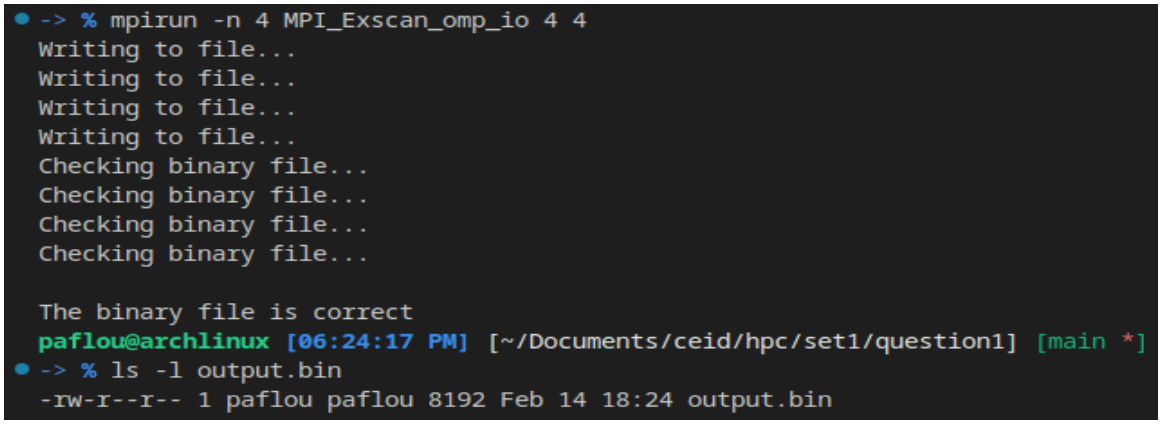
Με την κλήση της MPI_Exscan_omp αρχικοποιούνται τα τοπικά offsets κάθε νήματος (εφόσον το κάθε νήμα γράφει N*N*N δεν χρειαζόμαστε το for loop του προηγούμενου ερωτήματος). Στην συνέχεια με τον ίδιο τρόπο με το προηγούμενο υποερώτημα υπολογίζεται το τελικό offset.

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 και MPI_File_read_at, καθώς καλούνται απο πολλά νήματα ταυτόχρονα και με χρήση των collective operations το πρόγραμμα συχνά κολλούσε.

Παράδειγμα εκτέλεσης



4) 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 και αποθηκεύεται σε καταλληλη μεταβλητη.

Η μόνη διαφορά στο MPI_Exscan_omp_io συγκριτικά με το προηγούμενο ερώτημα ειναι οτι επαναφέραμε τον κοινόχρηστο πίνακα για τις τιμές των νημάτων, καθώς δεν γράφουν πλέον όλα ακριβώς τον ίδιο αριθμό δεδομένων.

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.

Παράδειγμα εκτέλεσης

```
paflou@archlinux [06:24:19 PM] [~/Documents/ceid/hpc/set1/question1] [main *]
-> % mpirun -n 4 MPI_Exscan_omp_io_compressed 4 4
Writing to file...
Writing to file...
Writing to file...
Writing to file...

The binary file is correct
paflou@archlinux [06:24:41 PM] [~/Documents/ceid/hpc/set1/question1] [main *]
-> % ls -l output.bin
-rw-r--r-- 1 paflou paflou 8368 Feb 14 18:24 output.bin
```

Στο συγκεκριμένο παράδειγμα το αρχείο ειναι μεγαλυτερο(!) απο το μη συμπιεσμένο, λογω της ελλειπούς δυνατότητας συμπίεσης σε συνδυασμό με τον επιπλέον κώδικα που απαιτείται απο την zlib.

2. Parallel Parametric Search in Machine Learning

1) q2a.py

   Στον κώδικα που μας δώθηκε χρησιμοποιήσαμε την κλάση pool της βιβλιοθήκης multiprocessing.

   Μετατρέψαμε το for loop σε συνάρτηση και την καλέσαμε παράλληλα σε ενα pool διεργασιών.

2) q2b.py

   Στον κώδικα του προηγούμενου υποερωτήματος χρησιμοποιήσαμε την κλάση MPICommExecutor για να καλέσουμε την συνάρτηση evaluate.

   Το αρνητικό αυτής της υλοποίησης ειναι οτι ένα νήμα δεν εκτελεί εργασία, και απλώς συντονίζει τα υπολοιπα.

3) q2c.py

   Για αυτό το υποερώτημα δημιουργήσαμε 2 συναρτήσεις, master, worker.

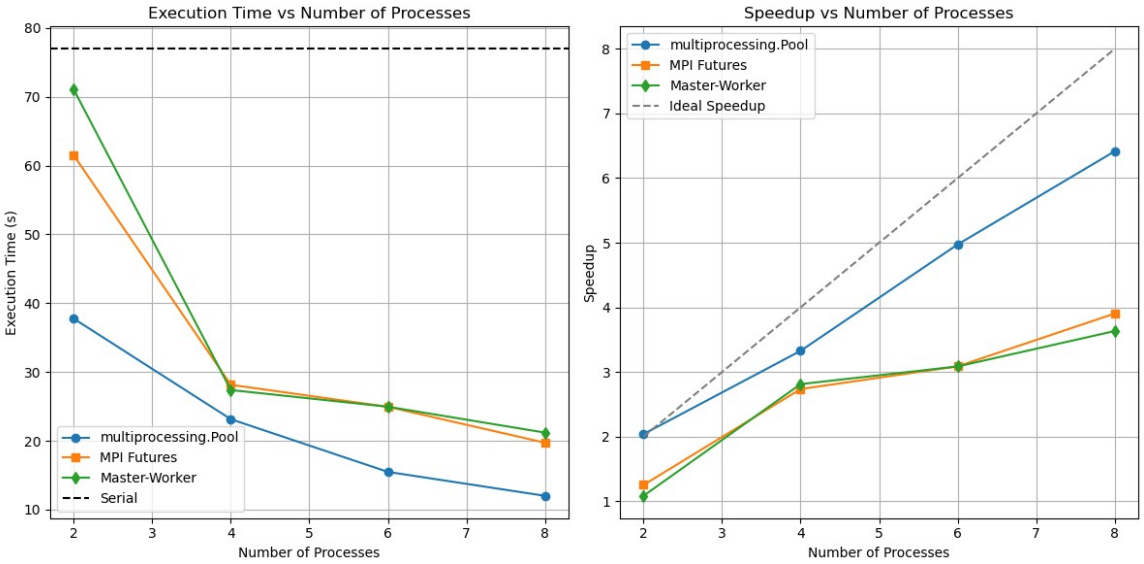
   Η διεργασία master στέλνει σε όλες τις διεργασίες worker τα iterations που πρέπει να υλοποιήσει η καθεμία και έπειτα τους στέλνει σήμα διακοπής. Τέλος παραλαμβάνει τα δεδομένα που έχουν στείλει οι εργάτες.

   Οι διεργασίες worker παραλαμβάνουν τα iterations ένα ένα, τα εκτελούν και τα στέλνουν πίσω στον master (ο οποίος, καθώς δεν εκτελεί δουλεία πέραν του διαμοιρασμού, ειναι έτοιμος για παραλαβή). Οταν λάβει συγκεκριμένο μήνυμα που σημαίνει οτι δεν έχει άλλη εργασία, τερματίζει.

   Το αρνητικό αυτής της υλοποίησης, όπως την προηγούμενη, είναι οτι ένα νήμα δεν εκτελεί εργασία, και απλώς συντονίζει τα υπολοιπα.

Μετρήσεις

Παράδειγμα εκτέλεσης με 90000 samples και 2 features



Τρέξαμε και τις 3 υλοποιήσεις με 90000 samples και 2 features (data.csv) και επιβεβαιώθηκε οτι η καλύτερη υλοποίηση ήταν της βιβλιοθήκης multiprocessing, με χρονοβελτίωση για **8 διεργασίες ~6.5** σε σύγκριση με το MPI_Futures με χρονοβελτίωση **~3.9** και το Master-Worker με **~3.7.**

Αξίζει να σημειωθεί οτι για 2 διεργασίες τα μοντελα futures και master-worker ήταν ανούσια, καθώς η 1 απο τις δυο διεργασίες δεν εκτελούσε τις πράξεις που χρειαζόμασταν, οπότε πρακτικά έγινε σειριακά (εξού και η χαμηλη χρονοβελτίωση για N=2).

3. OpenMP Tasking

A) 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. *N0 -> 0,1 | N1 -> 2,3 | N2 -> 4,5 | N3→6,7*

και όποτε κάποιο νήμα ολοκληρώνει τις επαναλήψεις του του ανατείθενται οι 2 επόμενες δυναμικά *(εαν το N2 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**.