Παράλληλα Συστήματα
Εργασία MPI 2018-2019:
Προσομοίωση μεταφοράς θερμότητας

Μαραγκοζάκης Γεώργιος – 1115201500089
Σπίθας Ευάγγελος – 115201500147

Περιεχόμενα

1. Γενικός Σχεδιασμός
2. Επεκτάσεις
3. Επιλογές κώδικα MPI
4. Μετρήσεις – Μελέτη Κλιμάκωσης
5. Συγκρίσεις υλοποιήσεων
6. Σχόλια – Εντολές εκτέλεσης

Γενικός Σχεδιασμός

- **Επιλογή Διεργασιών**: Αρχικά στην υλοποίηση μας, ελέγχεται αν το πλήθος διεργασιών (tasks) που έχει δοθεί από την γραμμή εντολής είναι ικανό να διαιρεθεί ακριβώς με το πλήθος των δεδομένων του αρχείου των θερμοκρασιών. Στην περίπτωση που δεν υπάρχει τέλεια διαίρεση, τότε μειώνεται κατά 1 το πλήθος των διεργασιών (tasks) και επαναλαμβάνεται ο παραπάνω έλεγχος. Η διαδικασία αυτή ακολουθείται για συγκεκριμένα πλήθη διεργασιών όπως: 9, 25, 36, 49, 81, 100, 121, 144. Για παράδειγμα, για tasks = 9 το πραγματικό πλήθος διεργασιών που συμμετέχει στο MPI είναι 8. Σε κάθε περίπτωση οι επιπλέον διεργασίες που περισσεύουν δεν πραγματοποιούν καμία διεργασία, αλλά διατρέχουν το πρόγραμμα και τερματίζουν.
- **Διάσπαση Αριθμού Διεργασιών σε γινόμενο Γραμμών – Στηλών (Διεργασιών)**: Το νούμερο των διεργασιών (tasks), διασπάται σε δύο νούμερα διεργασιών που αφορούν τις διεργασίες ανά γραμμές και τις διεργασίες ανά στήλες του αρχείου-πίνακα θερμοκρασιών. Το γινόμενο των (sqrt_tasks, sqrt_tasks2) πρέπει να δίνει τον αριθμό των διεργασιών (tasks). Για να υπολογιστούν οι 2 αυτές τιμές, αρχικά υπολογίζεται η τετραγωνική ρίζα (sqrt_taks) του συνολικού αριθμού των tasks. Στη συνέχεια, υπολογίζεται το πηλίκο της διαίρεσης(sqrt_tasks2) του αριθμού των tasks με την τετραγωνική του ρίζα. Σε περίπτωση που το tasks δεν έχει ακέραια τετραγωνική ρίζα τότε το γινόμενο sqrt_tasks * sqrt_tasks2 είναι μικρότερο από το νούμερο των tasks. Για το λόγο αυτό το υπόλοιπο (rest) της προηγούμενης διαίρεσης tasks / sqrt_tasks, είναι μεγαλύτερο του 0. Για να αντιμετωπιστεί αυτό, μέσα σε μια επαναληπτική διαδικασία, αυξάνεται ή μειώνεται το sqrt_tasks, ώστε να υπάρχει τέλεια διαίρεση με τον αριθμό των tasks και να ικανοποιείται η παραπάνω συνθήκη για μηδενικό υπόλοιπο. Η επιλογή για την αύξηση ή μείωση του sqrt_tasks γίνεται με τον εξής τρόπο: αν η τετραγωνική ρίζα (sqrt_tasks) είναι μεγαλύτερη από το μισό του αριθμού των tasks τότε μείωσε την κατά 1 και επανέλαβε τη διαίρεση. Αντίστοιχα, αν είναι μικρότερη, αύξησε την κατά 1. Τα αποτελέσματα αυτής της διαδικασίας είναι 2 νούμερα sqrt_tasks, και sqrt_tasks2, τα οποία διαιρούν το πλήθος των διεργασιών σε πλήθος διεργασιών ανά γραμμή και ανά στήλη. Ένα γραφικό παράδειγμα του παραπάνω απεικονίζεται εντός του φακέλου PM και στο αρχείο table. Στο παρακάτω παράδειγμα για **tasks = 49**, το οποίο γίνεται **40** για να διαιρείται ακριβώς το αρχείο, υπάρχουν **5 γραμμές** και **8 στήλες**, τα οποία νούμερα αναπαριστούν τις διεργασίες ανά γραμμή και ανά στήλη.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

- **Αριθμός γραμμών και στηλών δεδομένων ανά διεργασία**: Αφού υπολογίστηκαν τα sqrt_tasks, και sqrt_tasks2, τώρα υπολογίζονται τα δεδομένα ανά γραμμή και ανά στήλη του μπλοκ δεδομένων κάθε διεργασίας. Η υπόθεση είναι ότι το αρχείο διασπάται σε ίδιου μεγέθους μπλοκ δεδομένων. Ανάλογα ποιο από τα δύο νούμερα sqrt_tasks, και sqrt_tasks2, διαιρεί ακριβώς το μέγεθος της αρχικής γραμμής και στήλης του αρχείου δεδομένων υπολογίζονται τα num_x και num_y. Στη συνέχεια, δημιουργείται ο πίνακας τύπου struct, row_column, θέσεων όσες και οι διεργασίες (tasks) που συμμετέχουν πρακτικά στο πρόγραμμα. Σε κάθε θέση του πίνακα αποθηκεύονται το νούμερο (i_x) στο οποίο ξεκινά η γραμμή του μπλοκ δεδομένων κάθε διεργασίας, καθώς και το νούμερο (i_y) στο οποίο ξεκινά η στήλη του μπλοκ δεδομένων της συγκεκριμένης διεργασίας. Τέλος , αποθηκεύεται και το πλήθος των δεδομένων του πίνακα – μπλοκ δεδομένων κάθε διεργασίας.
- **Δημιουργία πινάκων μπλοκ – δεδομένων διεργασιών**: Η υπόθεση εδώ είναι ότι κάθε διεργασία έχει έναν πίνακα για το μπλοκ δεδομένων που της αναλογεί καθώς και 2 γραμμές και 2 στήλες επιπλέον για να λαμβάνει δεδομένα από τις γειτονικές της διεργασίες, (βόρεια, νότια, δύση, ανατολή).
  Για παράδειγμα, για μια διεργασία με μπλοκ δεδομένων x * y, δημιουργείται ένας πίνακας μεγέθους (x+2)*(y+2). Στα εσωτερικά άσπρα πεδία τοποθετούνται τα δεδομένα του μπλοκ, ενώ στα κίτρινα οι εισερχόμενες γραμμές και στήλες δεδομένων από τους γείτονες.

	N	N		N	N	
W			. . .			E
W			…			E
W						E
	S	S		S	S	

Η παραπάνω υλοποίηση έγινε, για να μην χρησιμοποιούνται ενδιάμεσοι buffer για την προσωρινή αποθήκευση των εισερχομένων δεδομένων από τις γειτονικές διεργασίες.
Στο παραπάνω σχεδιάγραμμα, για λόγους κατανόησης, ο πίνακας αναπαρίσταται ως δυο διαστάσεων. Στην υλοποίηση όμως του προγράμματος, ο πίνακας είναι μιας διάστασης.

- **Αρχικοποίηση αρχείου δεδομένων**: Η διεργασία 0 (Master), αναλαμβάνει να δημιουργήσει και να αρχικοποιήσει το αρχείο δεδομένων από το οποίο θα μοιραστούν τα δεδομένα σε μορφή μπλοκ όλες οι διεργασίες. Για λόγους διευκόλυνσης του parallel I/O, εξηγείται παρακάτω, το αρχείο δημιουργείται και σε μορφή .bin.
- **Υπολογισμός χρόνου**: Ακριβώς πριν το for – loop για αριθμό από STEPS, υπολογίζεται ο χρόνος εκείνη τη στιγμή (startTime) και ακριβώς μετά το for-loop, ο χρόνος του endTime. Στη συνέχεια, η κάθε μια από τις διεργασίες που συμμετέχουν πρακτικά στο πρόγραμμα στέλνει (MPI_SEND) το χρόνο της στην διεργασία 0. Η διεργασία 0 αναλαμβάνει να μαζέψει τους χρόνους από όλες τις άλλες διεργασίες (MPI_RECV), τόσες φορές όσες και οι διεργασίες. Τέλος υπολογίζει το maxtime, το mintime, καθώς και το averagetime για αυτήν την εκτέλεση.

Επεκτάσεις

- **Parallel I/O**: Για τον διαχωρισμό του αρχικού αρχείου δεδομένων σε μπλοκ – δεδομένων ανά διεργασία, επιλέχθηκε η παράλληλη ανάγνωση - γραφή. Η λογική πίσω από την χρησιμοποίηση του είναι ότι κάθε διεργασία, ανοίγει το αρχικό αρχείο, τοποθετεί στην διεύθυνση της μεταβλητής MPI_FILE τη διεύθυνση του πρώτου byte του αρχείου. Στη συνέχεια εντός ενός for-loop, για κάθε γραμμή του μπλοκ δεδομένων, τροποποιείται η διεύθυνση της μεταβλητής διαβάσματος του αρχείου, και μέσω της MPI_FILE_SEEK. Έτσι, λοιπόν, η διεύθυνση της μεταβλητής διαβάσματος δείχνει στο συγκεκριμένο byte της γραμμής του αρχείου .Στη συνέχεια, ο πίνακας του μπλοκ που αναφέρθηκε παραπάνω δείχνει μέσω ενός pointer στη θέση στην οποία θα διαβαστούν numy δεδομένα από την συγκεκριμένη γραμμή του αρχείου. Τέλος, η MPI_FILE_READ, διαβάζει num_y δεδομένα από την γραμμή του αρχείου. Η ίδια διαδικασία, επαναλαμβάνεται για όσες γραμμές του αρχείου πρέπει να τοποθετηθούν πλήθος στηλών στον πίνακα – μπλοκ. Αντίστοιχα στο MPI_FILE_WRITE, για κάθε διεργασία και κάθε γραμμή αυτής η μεταβλητή του δείκτη του αρχείου δείχνει στο αντίστοιχο byte του αρχείου και γράφονται τα δεδομένα κάθε γραμμής του μπλοκ δεδομένων της διεργασίας. **Η υλοποίηση του Parallel I/O έγινε έγινε λαμβάνοντας υπόψιν σημειώσεις και παραδείγματα από εδώ στον ιστότοπο (διαφάνεια 26).**

- Στην επόμενη σελίδα ακολουθεί γραφικό παράδειγμα

- Παράδειγμα για αρχείο μεγέθους **X = 10** και **Y = 8**, με **num_x = 5** και **numy = 4** για συνολικό αριθμό διεργασιών **tasks = 4**.
  - Αρχικό αρχείο μεγέθους 10 * 8
    - Διεργασία 0 -> πράσινα Διεργασία 1 -> μπλε
    - Διεργασία 2 -> κόκκινα Διεργασία 3 -> πορτοκαλί

0	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							
7							
8							
9							

  - Πίνακας μπλοκ δεδομένων **διεργασίας 0** (τα κίτρινα κελιά, όπως ειπώθηκε παραπάνω αφορούν τα εισερχόμενα δεδομένα από γειτονικές διεργασίες)

0	1	2	3	4	5
1	0,0	0,1	0,1	0,3	
2	1,0	1,1	1,2	1,3	
3	2,0	2,1	2,2	2,3	
4	3,0	3,1	3,2	3,3	
5	4,0	4,1	4,2	4,3	
6					

    - Στην 1η επανάληψη, μεταβλητή αρχείου -> 0,0 και διαβάζονται 4 δεδομένα, μέχρι το 0,3
    - Στην 2η >>, μεταβλητή αρχείου -> 1,0 και διαβάζονται 4 δεδομένα, μέχρι το 1,3
    - Στην 3η >>, μεταβλητή αρχείου -> 2,0 και >> 4 δεδομένα, μέχρι το 2,3
    - Στην 4η >>, μεταβλητή αρχείου -> 3,0 και >> 4 δεδομένα, μέχρι το 3,3
    - Στην 5η >>, μεταβλητή αρχείου -> 4,0 και >> 4 δεδομένα, μέχρι το 4,3

Επιλογές κώδικα MPI

- MPI:
  - **MPI_Datatypes για γραμμές και στήλες**: Για τις γραμμές χρησιμοποιείται MPI_Type_contiguous με μέγεθος num_y δεδομένων, ενώ για τις στήλες ένα MPI_Type_vector μεγέθους num_x και με απόσταση ανάμεσα σε κάθε δεδομένο από τον πίνακα μπλοκ, num_y+2, και με αρχική γραμμή την 1η του πίνακα μπλοκ.
    Για παράδειγμα όπως στο προηγούμενο για το Parallel I/O
    Πίνακας μπλοκ δεδομένων

0	1	2	3	4	5
1					
2					
3					
4					
5					
6					

    - Γραμμή με ΜΠΛΕ κελιά, δημιουργείται με MPI_Type_contiguous
    - Στήλη με ΜΩΒ κελιά, δημιουργείται με MPI_Type_vector
  - **Αρχικοποίηση γειτονικών διεργασιών**: Σε κάθε διεργασία, οι γείτονες της αρχικοποιούνται όλοι στο MPI_PROC_NULL. Στη συνέχεια, υπολογίζεται εκ νέου ο αριθμός των διεργασιών στηλών και με βάση των αριθμό της τωρινής διεργασίας, υπολογίζονται οι γειτονικές διεργασίες. Όσες διεργασίες είναι στα όρια του πίνακα διεργασιών (βλ. Αρχείο table.txt), παραμένουν με γείτονες το MPI_PROC_NULL. Οι γείτονες για κάθε διεργασία ,αποθηκεύονται στον πίνακα struct row_column, με index taskid για κάθε διεργασία.
    Για παράδειγμα, ο πίνακας διεργασιών για **40 διεργασίες** όπως ειπώθηκε και σε παραπάνω παράδειγμα.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

    Ο αριθμός των διεργασιών στηλών είναι temp_sqrt = 8.
    - Διεργασία 1, (taskid = 0)

      UP = MPI_PROC_NULL

      LEFT = MPI_PROC_NULL RIGHT = taskid + 1 = 1

      DOWN = taskid + temp_sqrt = 8

- **Αποστολή οριακών γραμμών στηλών**: Σε κάθε διεργασία, στέλνονται με την βοήθεια των παραπάνω MPI_Datatypes, οι 2 οριακές γραμμές και οι 2 οριακές στήλες δεδομένων του μπλοκ, στις αντίστοιχες γειτονικές διεργασίες. Για να πραγματοποιηθεί αυτό γίνεται με τη χρήση **4 x Isend**. Αντίστοιχα, σε κάθε διεργασία εκτελούνται **4 x Recv** για τη λήψη των δεδομένων από τις γειτονικές διεργασίες, και τοποθετούνται στις οριακές(κίτρινες) θέσεις του πίνακα μπλοκ δεδομένων, όπως αυτός παρουσιάστηκε σε προηγούμενο παράδειγμα.
- **Υπολογισμός εσωτερικών στοιχείων πίνακα μπλοκ δεδομένων**: Εφόσον, τα Isend και Irecv, πραγματοποιούνται ασύγχρονα, καλείται η συνάρτηση update για τον υπολογισμό των νέων θερμοκρασιών των εσωτερικών δεδομένων του πίνακα μπλοκ – δεδομένων της κάθε διεργασίας.
  Παράδειγμα εσωτερικών στοιχείων του πίνακα μπλοκ δεδομένων.

0	1	2	3	4	5
1					
2					
3					
4					
5					
6					

  *Τα πράσινα κελιά είναι τα εσωτερικά στοιχεία πίνακα – μπλοκ δεδομένων
- **Υπολογισμός εξωτερικών στοιχείων πίνακα μπλοκ δεδομένων**: Αφού εκτελεστούν επιτυχώς **4 x MPI_Wait** για τα παραπάνω Irecv, ώστε να έχουν ληφθεί τα δεδομένα από τις γειτονικές διεργασίες, γίνονται νέες κλήσεις **4 x update**, για τις δυο εξωτερικές γραμμές και στήλες του πίνακα μπλοκ δεδομένων.
  *Τα μωβ κελιά είναι τα εξωτερικά στοιχεία πίνακα – μπλοκ δεδομένων(βλ. Παράδειγμα)
- **Αναμονή για Senders**: Τέλος, πριν συνεχίσει στο επόμενο βήμα του for-loop, καλούνται 4 x MPI_Wait, μέχρι να αποσταλούν επιτυχώς στις γειτονικές διεργασίες τα οριακά δεδομένα του πίνακα.

- MPI + Reduce:
  - **Έλεγχος για αλλαγή δεδομένων**: Εντός της συνάρτησης update, μόλις υπολογιστεί η νέα τιμή κάθε κελιού του πίνακα – μπλοκ δεδομένων της κάθε διεργασίας, ελέγχεται η νέα τιμή του κελιού του πίνακα u2 με την παλιά τιμή του κελιού του πίνακα u1. Αν οι τιμές είναι διαφορετικές τότε η global μεταβλητή change_column αυξάνεται κατά μία μονάδα.
  - **Κλήση MPI_Allreduce**: Όλες οι διεργασίες, είτε συμμετέχουν πρακτικά είτε όχι, στο πρόγραμμα, καλούν την **MPI_Allreduce**, με τα αντίστοιχα ορίσματα και τη global μεταβλητή change_column. Το αποτέλεσμα της κλήσης της συνάρτησης επιστρέφεται στο result_column.

- MPI + Persistent Communication:
  - **Αλλαγές στον κώδικα MPI**: Αντικαταστάθηκαν οι εντολές Isend και Irecv με τις εντολές Send_Init και Recv_Init εκτός του κεντρικού loop, και εντός του loop κλήθηκε η Start για κάθε request το οποίο αφορούσε τις 4 αποστολές σε γειτονικές διεργασίες και τις 4 λήψεις από αυτές. Υλοποιήθηκε ξεχωριστά για να γίνει σύγκριση έναντι του απλού MPI.
- MPI + Reduce + OpenMp:
  - Για το κομμάτι του MPI χρησιμοποιήσαμε την εντολή ”#pragma omp parallel for” μέσα στην συνάρτηση update έξω από το βασικό loop το οποίο πραγματοποιεί όλους τους υπολογισμούς. Ο λόγος που το κάναμε αυτό ήταν γιατί θέλαμε να παραλληλοποιήσουμε μόνο αυτό το κομμάτι αφού πέρα από αυτό δεν πραγματοποιούμε άλλους υπολογισμούς παρά μόνο send/receive καθώς και ένα reduce.
  - Για την παραλληλοποίηση με Threads στην update είχαμε 2 επιλογές. Είτε να χρησιμοποιήσουμε το ”#pragma omp parallel for” είτε το ”#pragma omp parallel for collapse(2”). Η πρώτη εντολή παραλληλοποιεί μόνο το εξωτερικό loop, δηλαδή αν έχουμε δηλώσει 4 threads και έχουμε ένα loop ”for i=0; i<8; i++” τότε θα έχουμε συνολικά 8/4=2 εκτελέσεις του loop σε κάθε thread. Αντίθετα αν είχαμε χρησιμοποιήσει την 2η εντολή τότε η παραλληλοποίηση θα γίνονταν και στα 2 loop.
    - Δηλαδή αν είχαμε: for i=0; i<8; i++
      for j=0; j<2; j++

      θα είχαμε σύνολο 2*8=16 επαναλήψεις και αυτές θα χωρίζονταν στα Threads. Δηλαδή θα είχαμε 16/4=4 εκτελέσεις ανά thread. Ο λόγος που διαλέξαμε την πρώτη επιλογή είναι επειδή κάναμε κάποιες δοκιμές με μικρά παραδείγματα και είδαμε ότι είχε γρηγορότερους χρόνους εκτέλεσης.
  - Τέλος μέσα στο loop της update χρησιμοποιήσαμε την εντολή ”#pragma openmp critical” έξω από το memcmp για να διασφαλίσουμε ότι κάθε thread θα πραγματοποιήσει σωστά τον έλεγχο χωρίς να επηρεάσει τα δεδομένα των υπόλοιπων.

Μετρήσεις – Μελέτη Κλιμάκωσης

- **Μετρήσεις στο rbs.marie.hellasgrid.gr**: Οι μετρήσεις βρίσκονται στο φάκελο timings. Το πρόγραμμα εκτελέστηκε για αριθμό διεργασιών:
  - 1, 4, 9, 16 25, 36, 49, 64, 81, 100, 121, 128, 144, 160
    και για κάθε έναν από τον αριθμό αυτών των διεργασιών με μεγέθη αρχείου:
  - 80x64, 160x128, 320x256, 640x1024
    Σε ορισμένα μεγάλα μεγέθη διεργασιών η εκτέλεση τους στο εργαστήριο rbs δεν ήταν πάντα επιτυχής, καθώς ειδικά όταν υπήρχε φόρτος στο σύστημα εμφανιζόταν το μήνυμα "ORTE has lost communication with a remote daemon...", γι' αυτό και έγιναν πολλές προσπάθειες για την επιτυχή εκτέλεση τους.
  - Όλες οι μετρήσεις έγιναν με STEPS = 100.
  - Οι χρόνοι των μετρήσεων είναι ο μέσος χρόνος όλων των διεργασιών καθώς και μέγιστος και ελάχιστος χρόνος από τις διεργασίες.
  - Σε ορισμένα μεγέθη αρχείου, κυρίως μικρά, ανεβάζοντας τον αριθμό των διεργασιών από ένα όριο και πάνω οι επιδόσεις τους ήταν χειρότερες από τη σειριακή για αυτό και δεν πάρθηκαν μετρήσεις για αυτές.
  - Στις μετρήσεις που αφορούν το OpenMp από κάθε αριθμό tasks κρατήθηκαν στη γραφική παράσταση της επιτάχυνσης και της αποδοτικότητας οι συνδυασμοί tasks – threads με τις καλύτερες επιδόσεις.
  - Οι μετρήσεις πραγματοποιήθηκαν το διάστημα 23-26 Σεπτέμβρη και αρκετές από αυτές σε ώρες αιχμής του εργαστηρίου. Για το λόγο αυτό ορισμένα νούμερα αποκλίνουν από τα αναμενόμενα.

- **Επιτάχυνση και Αποδοτικότητα**:
  - Επιτάχυνση: S = T_σειριακό / T_παράλληλο
  - Αποδοτικότητα: E = S / p
  - Στο αρχείο graphs βρίσκονται οι γραφικές παραστάσεις για την επιτάχυνση και την αποδοτικότητα για όλες τις μετρήσεις μας.

- **Μελέτη Κλιμάκωσης**:
  - **Μικρά μεγέθη αρχείων** Ο χρόνος εκτέλεσης μειώνεται μέχρι ένα μέγεθος διεργασιών, και στη συνέχεια, βλέπουμε ότι αυξάνεται και φτάνοντας στο ταβάνι των 160 διεργασιών παρατηρείται χειρότερος χρόνος επίδοσης από αυτόν του σειριακού προγράμματος. Αυτό συμβαίνει καθώς το μικρό πλήθος δεδομένων επιβαρύνεται με το μεγάλο πλήθος επικοινωνιών μεταξύ των διεργασιών. Επιπλέον, για αυτά τα μεγέθη αρχείων η επιτάχυνση είναι μικρότερη από τις πρώτες κιόλας μετρήσεις. Αντίστοιχα η αποδοτικότητα ακολουθεί καθοδική πορεία για αυτά τα μεγέθη από νωρίς. Αυτό ερμηνεύεται ως ότι οι δυνατότητες εκτέλεσης εντολών από έναν πυρήνα είναι αρκετά υψηλές σε μικρό πλήθος δεδομένων.

- **Μεγάλα μεγέθη αρχείων**: Ο χρόνος εκτέλεσης για τα μεγάλα μεγέθη αρχείων (640χ1024), μειωνόταν όσο αυξάνονταν το πλήθος των διεργασιών. Ακόμη και στις 160 διεργασίες ο χρόνος εκτέλεσης ήταν σαφώς καλύτερος από τον αρχικό, όμως η επιτάχυνση ήταν σαφώς πολύ μικρότερη. Παρατηρώντας και τα γραφήματα βλέπουμε ότι το πρόγραμμα μας είχε σαφώς καλύτερη κλιμάκωση στα μεγάλα μεγέθη του αρχείου.
- **Παρατηρήσεις για μετρήσεις**: Σε αρκετές μετρήσεις παρατηρείται ένα φαινόμενο, ειδικά σε μεγέθη διεργασιών που δεν διαιρούν ακριβώς το αρχείο και καταλήγουν στον ίδιο αριθμό διεργασιών (βλ. 81, 100, 121), διαφορές στις επιδόσεις. Αυτό προκύπτει, από τον διαφορετικό αριθμό κόμβων που επιλέγονται και το πως μοιράζονται οι διεργασίες στους κόμβους αυτούς, μιας και δεν έχει υλοποιηθεί η "Τοποθέτηση διεργασιών που επικοινωνούν στον ίδιο κόμβο".

Συγκρίσεις υλοποιήσεων

- **Συγκρίσεις MPI με MPI + Reduce**: Όπως παρατηρείται από τα γραφήματα των αποτελεσμάτων, στα μικρά μεγέθη οι διαφορές είναι ελάχιστες έως μηδαμινές. Όσο όμως αυξάνεται το μέγεθος του αρχείου βλέπουμε μια υπεροχή του MPI έναντι του MPI+Reduce. Αυτό συμβαίνει διότι και στα δύο οι πράξεις και οι επικοινωνίες των διεργασιών για τον υπολογισμό των δεδομένων είναι οι ίδιες. Όμως στην υλοποίηση MPI+Reduce σε κάθε επανάληψη υπάρχει μια επιπλέον επικοινωνία μεταξύ όλων των διεργασιών, μέσω της All_Reduce. Αυτό προσθέτει έναν εύλογο χρόνο στο πρόγραμμα και έτσι υστερεί σε επιδόσεις σε σχέση με το απλό MPI.
- **Συγκρίσεις MPI με MPI + Persistent Communication**: Οι διαφορές στις επιδόσεις γίνονται εμφανείς στα αρχεία μεγάλου μεγέθους, μιας και όπως φαίνεται από το γράφημα, η επιτάχυνση και κατ' επέκταση η αποδοτικότητα του Persistent Communication είναι σαφώς μεγαλύτερη και διατηρείται για μεγαλύτερο αριθμό διεργασιών από ότι του MPI. Αυτό συμβαίνει καθώς ο υπολογισμός των παραμέτρων για τα Isend και Irecv υπολογίζεται μια φορά εκτός της κεντρικής επανάληψης, με αποτέλεσμα να "κερδίζεται" χρόνος από επαναλαμβανόμενες πράξεις εντός της κεντρικής επανάληψης.
- **Συγκρίσεις MPI με OpenMp**: Οι μετρήσεις δείχνουν ότι όσο αυξάνεται το μέγεθος του αρχείου αλλά και οι διεργασίες ταυτόχρονα, φαίνεται μια υπεροχή του OpenMp έναντι του MPI, αλλά και μια διατήρηση της επιτάχυνσης του πρώτου σε ένα καλό επίπεδο, την ώρα που η επιτάχυνση του MPI μειώνεται δραστικά ειδικά από ένα όριο διεργασιών και πάνω. Επιπλέον, αν αναλογιστούμε ότι το OpenMp περιλαμβάνει και το reduce το οποίο είναι υπεύθυνο παραπάνω για μείωση των επιδόσεων, τότε το OpenMp κλιμακώνει πολύ καλύτερα έναντι του MPI.

- **Συγκρίσεις αρχικού προγράμματος με τωρινού**: Το αρχικό πρόγραμμα της εκφώνησης έκανε αρκετές παραπάνω πράξεις και κοστοβόρες, έναντι της υλοποίησης μας. Ορισμένες βελτιώσεις της υλοποίησης μας είναι:
  - Parallel I/O έναντι λογική MASTER process για καταμερισμό δεδομένων στις άλλες διεργασίες.
  - Χρήση μπλοκ δεδομένων έναντι γραμμών.
  - Χρήση Isend και Irecv για ασύγχρονες ανταλλαγές δεδομένων, χωρίς να κολλάει το πρόγραμμα μέχρι αυτές να ολοκληρωθούν.

- **Επεκτασιμότητα**
  - Το πρόγραμμα της υλοποίησης μας, είναι ασθενώς επεκτάσιμο καθώς αυξάνοντας το πλήθος των διεργασιών και ταυτόχρονα αυξάνοντας το μέγεθος του προβλήματος στο πρόγραμμα μας διατηρείται η αποδοτικότητα στα ίδια ποσοστά(βλ από 160x128 σε 320x256).
- **Σχόλια και Εντολές Εκτέλεσης**
  - Τροποποίηση συνάρτησης update: Η συνάρτηση update τροποποιήθηκε ως προς τα ορίσματα της για να επιτύχουμε τον αποτελεσματικότερο υπολογισμό των εξωτερικών σημείων του πίνακα μπλοκ δεδομένων. Για το λόγο αυτό στα ήδη υπάρχοντα ορίσματα της προστέθηκαν τα y_st και y_end για την αρχή και τέλος στις στήλες του πίνακα μπλοκ δεδομένων.
  - Μεταχείριση μη διαιρούμενων μεγεθών διεργασιών αρχείου: Όσες διεργασίες δεν διαιρούν ακριβώς το αρχείο, παίρνεται ένας μικρότερος αριθμός αυτών ο οποίος διαιρεί το αρχείο και οι υπόλοιπες δεν αναλαμβάνουν φόρτο μέσα στο πρόγραμμα. Επιπλέον, στο reduce αναλαμβάνουν μόνο να κάνουν μια κλήση all_reduce προς τις άλλες διεργασίες μην επηρεάζοντας το αποτέλεμσα.
  - Εντολές εκτέλεσης σε περιβάλλον linux (τοπικά)
    - Εντολή μεταγλώττισης και εκτέλεσης **./run.sh tasks X Y**,
      όπου tasks ο αριθμός των διεργασιών και X, Y τα μεγέθη του αρχείου(γραμμές, στήλες). Τα αποτελέσματα αποθηκεύονται μέσα στο φάκελο ./PM και η έξοδος του προγράμματος στο ./PM/out.txt
    - Για openMp προσθέτουμε ανάμεσα στα tasks και το X, τον αριθμό των threads
  - Εντολές για εκτέλεση σε περιβάλλον rbs
    - Εντολή μεταγλώττισης ./compile.sh
    - Εντολή εκτέλεσης για tasks που διαιρεί το αρχείο qsub myPBBScript.sh
    - Εντολή εκτέλεσης για tasks που δεν διαιρεί το αρχείο qsub myPBBScript2.sh
    - Σημείωση: πριν από κάθε εκτέλεση με το ./compile.sh διαγράφουμε τα περιεχόμενα του ./PM και τα ορίσματα δίνονται χειροκίνητα μέσα στα script από πριν