Συστήματα Παράλληλης Επεξεργασίας
Τέταρτη Εργαστηριακή Άσκηση

Αναστασία-Χριστίνα Λίβα 03119029
Γιώργος Μυστριώτης 03119065
Νικόλας Σταματόπουλος 03119020

Ιανουάριος 2024

Παραλληλοποίηση και βελτιστοποίηση αλγορίθμων σε αρχιτεκτονικές κατανεμημένης μνήμης

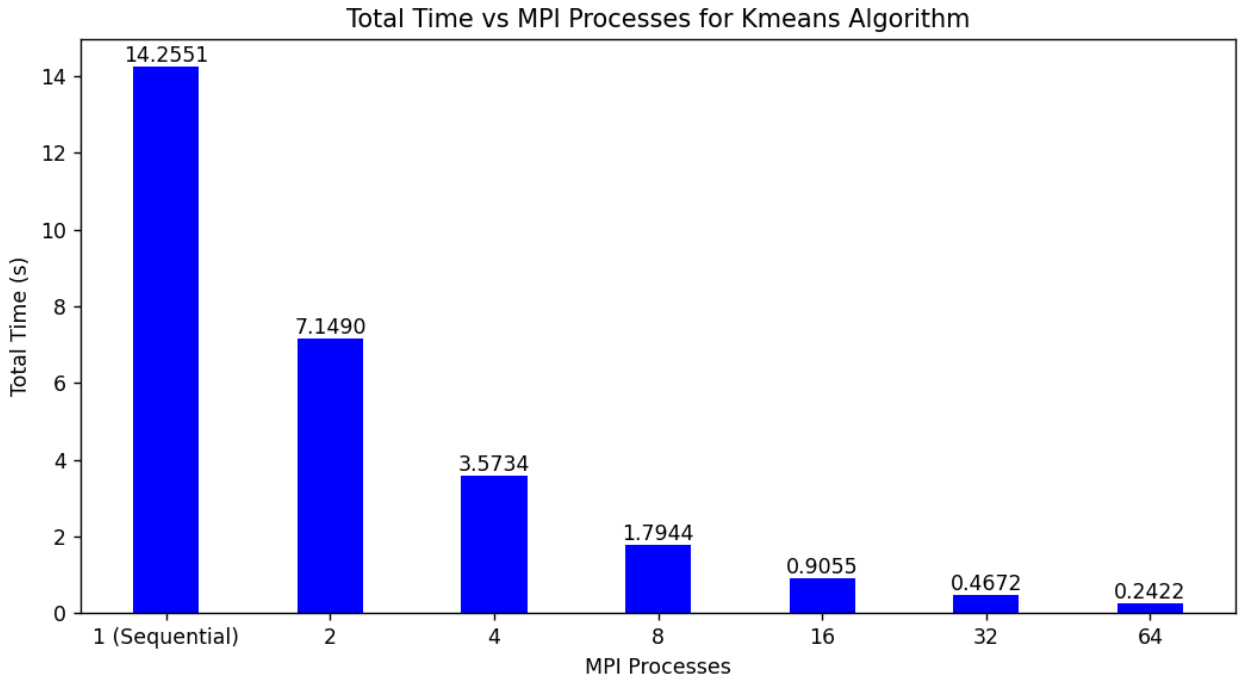
Άλλη μια παραλληλοποίηση και βελτιστοποίηση του αλγορίθμου K-means

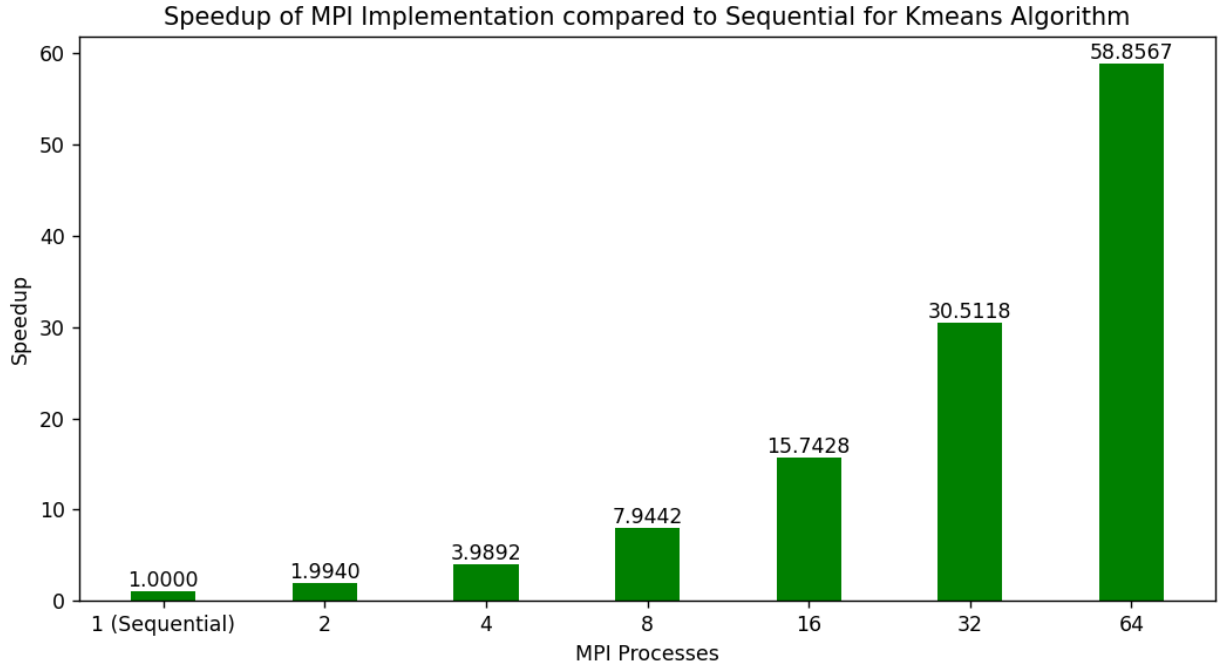
Διαμορφώνουμε αρχικά τα ζητούμενα κομάτια του κώδικα (TODOs), με σκοπό να εκτελέσουμε τον υπολογισμό των νέων clusters ανά τμήματα σε διεργασίες MPI. Έτσι παίρνουμε τα παρακάτω αποτελέσματα:

MPI Processes	Convergence (nloops)	Total Time (s)	Per Loop Time (s)
1	10	14.2551	1.4255
2	10	7.1490	0.7149
4	10	3.5734	0.3573
8	10	1.7944	0.1794
16	10	0.9055	0.0906
32	10	0.4672	0.0467
64	10	0.2422	0.0242

Πίνακας 1: Execution times for MPI processes, convergence, total time, and per loop time in the Kmeans algorithm (Size = 256(MB), numCoords = 16, numClusters = 16, Loops = 10).

Και επίσης παρουσιάζονται τα ζητούμενα διαγράμματα:





Η τελική διαμόρφωση του κώδικα φαίνεται στον φάκελο των παραδοτέων.

1. Όπως φαίνεται από τα αποτελέσματα σε κάθε αύξηση (x2) των MPI Process έχουμε και περίπου αντίστοιχη μείωση του per loop χρόνου και άρα και του συνολικού. Αυτό είναι λογικό, αφού ουσιαστικά μειώνουμε στο μισό το computation που απαιτείται από κάθε διεργασία, με μόνο επιπλέον overhead στον συνολικό χρόνο, το communication μεταξύ processes. Όμως, χρησιμοποιώντας functions είτε broadcast, είτε scatter είτε gather, δηλαδή functions που αφορούν όλο τον πληθυσμό των processes, το communication επιτυγχάνεται σε λογαριθμικό χρόνο (σε σχέση με το πλήθος διεργασιών) μέσω του optimizer του MPI. Συνεπώς χάνουμε αμελητέο χρόνο συγκριτικά με το computation που είναι τάξης $rank\_numObjs * numCoords$.

Μπορούμε προφανώς να δούμε τα πλεονεκτήματα στην MPI υλοποίηση, σε σχέση με την σειριακή, αφού το speedup είναι πολύ μεγαλό λόγω της παραλληλοποίησης. Το μόνο πρόβλημα που δημιουργείται είναι το overhead επικοινωνίας των διεργασιών, που όμως αντιμετωπίζεται με την χρήση κατάλληλων MPI functions, που οδηγούν το communication σε λογαριθμική τάξη (σε σχέση με το πλήθος διεργασιών), για να πάρουμε τα καλύτερα δυνατά αποτελέσματα.

Σύγκριση με OpenMP version: Οι μετρήσεις που είχαμε πάρει για την OpenMP reduction version ήταν:

| OpenMP Kmeans - Reduction (number of threads: 64)
| completed loop 10
| nloops = 10 (total = 0.2390s) (per loop = 0.0239s)

Θυμίζουμε πως για την MPI υλοποίηση είναι:

| MPI Kmeans (number of MPI processes: 64)
| nloops = 10 (total = 0.2422s) (per loop = 0.0242s)

Όπως φαίνεται, οι δύο υλοποιήσεις είναι πολύ κοντά σε χρόνους. Η κάθε μία χάνει κάποιο χρόνο σε διαφορετικά σημεία για να υπάρχει ακεραιότητα των δεδομένων. Δηλαδή, η μεν OpenMP θα πρέπει να προστατέψει κοινές μεταβλητές (π.χ. delta) και να εκτελέσει το reduction χωρίς παραλληλοποίηση, ενώ η MPI θα πρέπει να φροντίσει για την επικοινωνία των διεργασιών και επιπλέον να συνθέσει τα τελικά αποτελέσματα σε μια διεργασία (οπότε και αυτή χάνει ένα κομμάτι παραλληλοποίησης). Όπως βλέπουμε και από τους χρόνους εκτέλεσης, παρότι είναι πολύ κοντά, φαίνεται πως η επικοινωνία και αντιγραφή δεδομένων μεταξύ διεργασιών κοστίζει περισσότερο, από το reduction στην OpenMP έκδοση. Όμως, θα πρέπει να κρατήσουμε πως ένα MPI πρόγραμμα μπορεί να εκτελεστεί σε καταννεμημένα συστήματα με πολλούς ξεχωριστούς επεξεργαστές, οι οποίοι επικοινωνούν μέσω MPI.

Διάδοση θερμότητας σε δύο διαστάσεις

Ύστερα από την υλοποίηση των τριών μεθόδων σύμφωνα με την εκφώνηση, μπορούμε να τις συγκρίνουμε μεταξύ τους. Οι υλοποιήσεις βρίσκονται στον φάκελο των παραδοτέων.

Ερώτημα 1: Μετρήσεις με έλεγχο σύγκλισης

Για να κάνουμε έλεγχο σύγκλισης θα πρέπει στην αρχή των προγραμμάτων μας να ορίσουμε #define TEST_CONV. Παίρνοντας μετρήσεις για τα MPI προγράμματα μας με μέγεθος 1024 x 1024 και 64 MPI processes (grid 8x8) έχουμε:

Method	Size	Iteration	TotalTime	CommunicationTime	ComputationTime
Jacobi	1024 x 1024	798201	265.437	222.853086	40.744968
GaussSeidelSOR	1024 x 1024	3201	1.837	1.304837	0.475871
RedBlackSOR	1024 x 1024	2601	1.790	1.532253	0.239620

Πίνακας 2: Results for MPI runs of different algorithms solving heat transfer

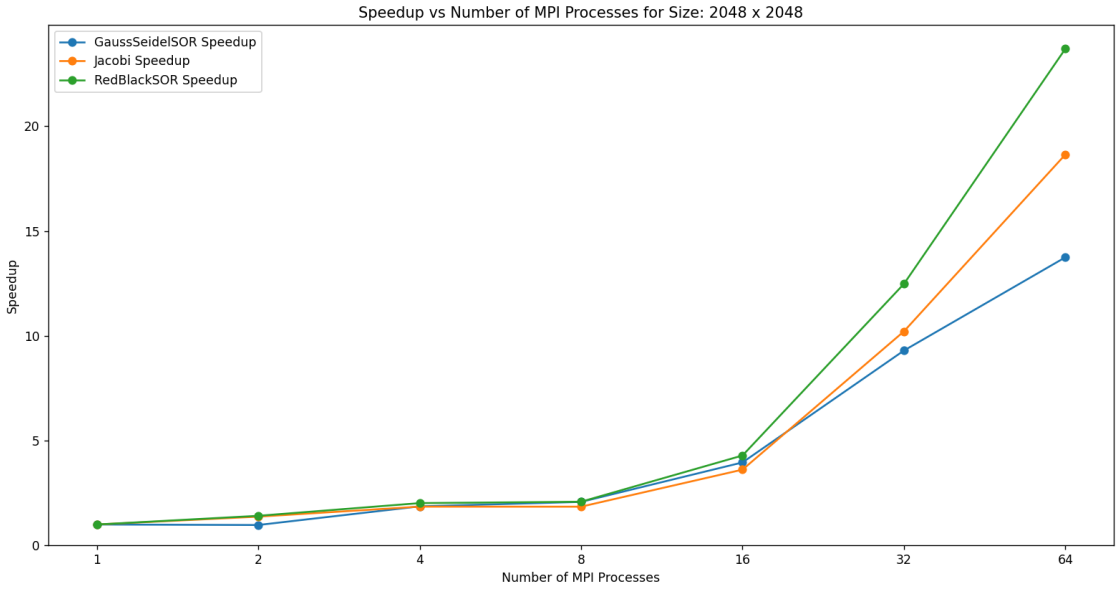
Όπως μπορούμε να δούμε η Μέθοδος RedBlackSOR μας δίνει το καλύτερο speedup, με την GaussSeidelSOR να είναι επίσης πολύ κοντά. Αντίθετα, μπορούμε να δούμε πως η μέθοδος Jacobi παρότι είναι απλή στην υλοποίηση, λόγω της αργής της σύγκλισης οδηγεί στα χειρότερο speedup. Ανάμεσα στις GaussSeidelSOR και RedBlackSOR αξίζει να σημειώσουμε κάποια πράγματα. Αρχικά, μπορούμε να παρατηρήσουμε πως παρότι το GaussSeidel έχει μεγαλύτερο communication time, κερδίζει αρκετά σε computation. Αυτό συμβαίνει επειδή στον αλγόριθμο αυτό, για τον υπολογισμό τους σημείου (i,j) θα πρέπει να προηγηθεί ο υπολογισμός των πάνω και αριστερά σημείων. Έτσι, χάνουμε σε κάποιο βαθμό την παραλληλοποίηση και πρέπει να περιμένουμε δεδομένα. Επιπλέον, μπορούμε να δούμε πως και οι δύο αλγόριθμοι συγκλίνουν ικανοποιητικά μειώνοντας πολύ τα iterations. Όμως, πρέπει να προσέξουμε πως για διαφορετικές τιμές του omega, βλέπουμε και άλλες ταχύτητες σύγκλισης. Τέλος, το καλύτερο speed up δίνει η RedBlackSOR, η οποία όπως βλέπουμε έχει την καλύτερη σύγκλιση και μικρό communication time ως ποσοστό του συνολικού χρόνου. Παρότι έχουμε να ανταλλάξουμε πολλά δεδομένα μεταξύ διεργασιών, επειδή πρέπει να στείλουμε rows και columns και του previous αλλά και του current πίνακα, το κύριο μέρος του total χρόνου αφορά το computation. Αν και δεν χάνουμε καθόλου σε παραλληλοποίηση (λόγω της μορφής του αλγορίθμου), η πολυπλοκότητά του οδηγεί σε μεγάλο computation time. Στην περίπτωσή μας όμως, που πετυχαίνει καλύτερη σύγκλιση, βλέπουμε μεγάλο speedup αφού μειώνεται ο αριθμός των επαναλήψεων και άρα και ο αριθμός υπολογισμών.

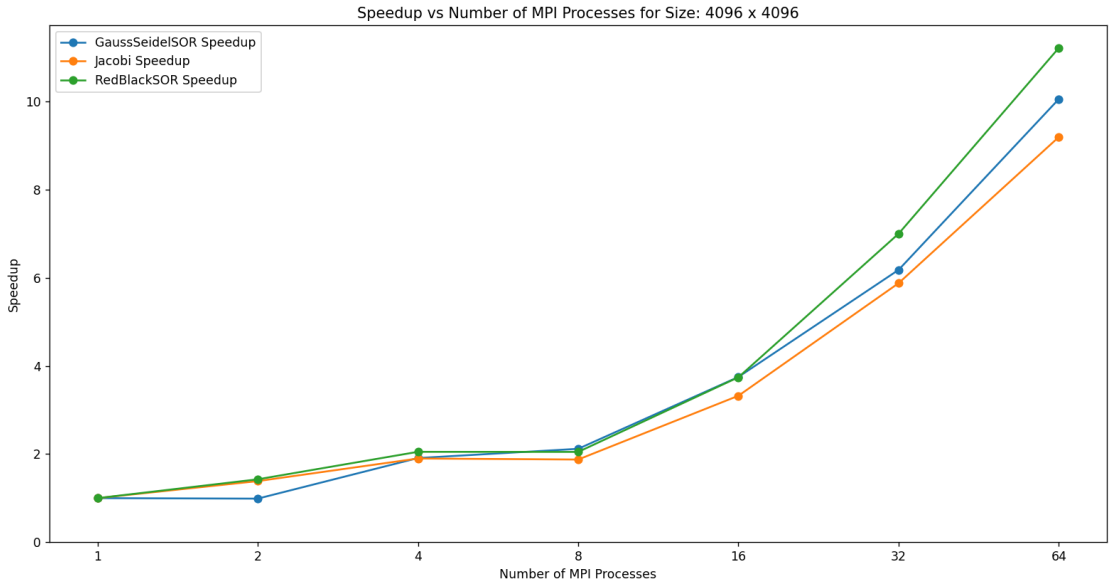
Συνεπώς, θα επιλέγαμε την μέθοδο GaussSeidelSOR, αφού έχουμε απλούστερη και γρηγορότερη εκτέλεση του αλγορίθμου και γρήγορη σύγκλιση. Το communication time μπορεί να αυξηθεί αρκετά λόγω της αλυσιδωτής φύσης του αλγορίθμου, αλλά είναι scalable, αφού κάθε διεργασία μιλάει πάντα με άλλες 4 και όχι με γραμμικά ή

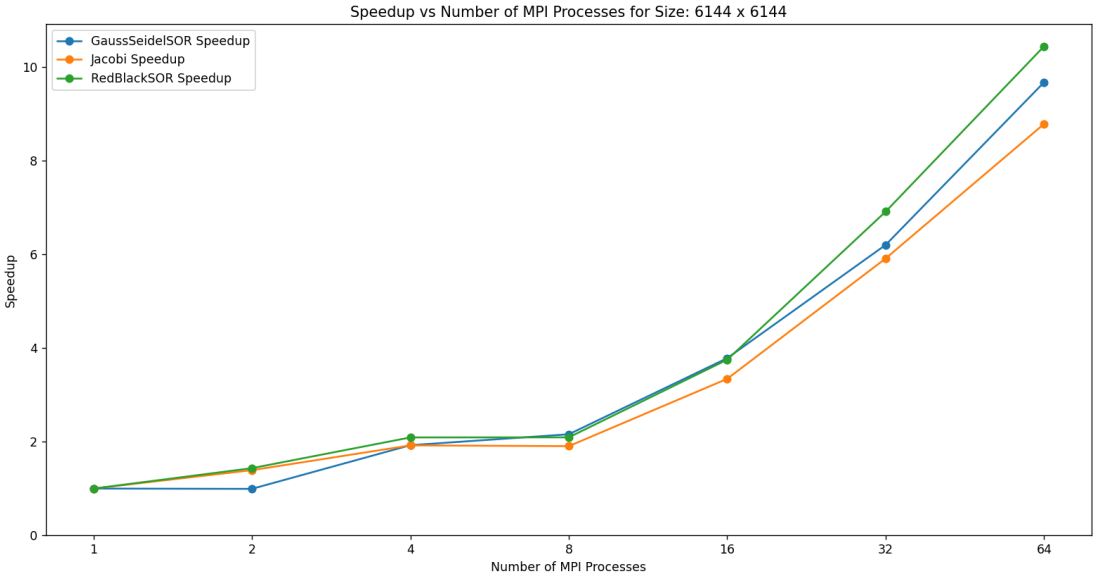
εκθετικά αυξανόμενο αριθμό. Επίσης, όπως είδαμε για μεγαλύτερα ταμπλό ο αλγόριθμος πλησιάζει το speedup του RedBlackSOR, λόγω του καλύτερού του scaleup.

Ερώτημα 2: Μετρήσεις χωρίς έλεγχο σύγκλισης

Κάνοντας undefine την TEST_CONV, μπορούμε πλέον για σταθερό T = 256 iterations να μελετήσουμε την συμπεριφορά των αλγορίθμων μας για διάφορες τιμές Size και MPI Processes. Έτσι παρουσιάζουμε τα ζητούμενα διαγράμματα:





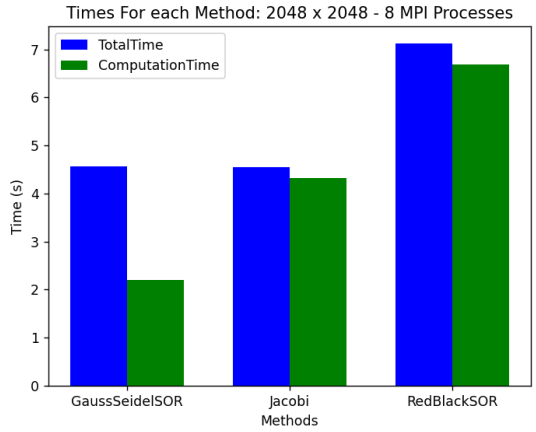


Όπως βλέπουμε όλες οι μέθοδοι έχουν καλό speedup, αφού οι αλγόριθμοι είναι scalable, δηλαδή η επικοινωνία για την ολοκλήρωση του αλγορίθμου αφορά σταθερό αριθμό διεργασιών, στην περίπτωσή μας 4, ενώ έχουμε ταυτόχρονα παραλληλοποίηση της διαδικασίας. Το καλύτερο speedup πετυχαίνει ο RedBlackSOR, ο οποίος με πλήρη παραλληλία και λόγω του scalability (από το οποίο θα υπέφερε αν δεν ήταν σταθερός ο αριθμός των διεργασιών που θα έπρεπε να επικοινωνήσει, λόγω του μεγάλου communication που απαιτεί) καταφέρνει να έχει το καλύτερο speedup για κάθε αριθμό MPI Porcesses. Παρολαυτά, λόγω των blocking Send και Recv, αλλά και του μεγαλύτερου κόστους επικοινωνίας, δεν κλιμακώνει καλά για μεγαλύτερες τιμές size του ταμπλό.

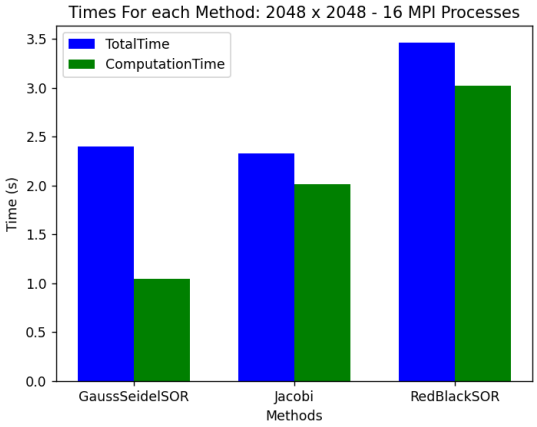
Αξίζει να σημειωθεί πως καλό scale up πετυχαίνει ο GaussSeidelSOR, ο οποίος για 2048 x 2048 πετυχαίνει μικρότερο speedup από τον Jacobi, αλλά για μεγαλύτερους πίνακες τον ξεπερνά και συνεχώς πλησιάζει και τον RedBlackSOR. Λόγω των non-blocking Isend και Irecv, μπορούμε να δούμε πως οι τιμές για το speedup του είναι σχετικά σταθερές για οποιοδήποτε μέγεθος πίνακα.

Τέλος, ο Jacobi έχει κακή κλιμάκωση αναφορικά με το μέγεθος του ταμπλό. Μπορούμε να δούμε πως για μικρό μέγεθος 2048 x 2048, ξεπερνά τον GaussSeidel, αλλά για μεγαλύτερες τιμές λόγω των blocking Send και Recv χάνει σε απόδοση.

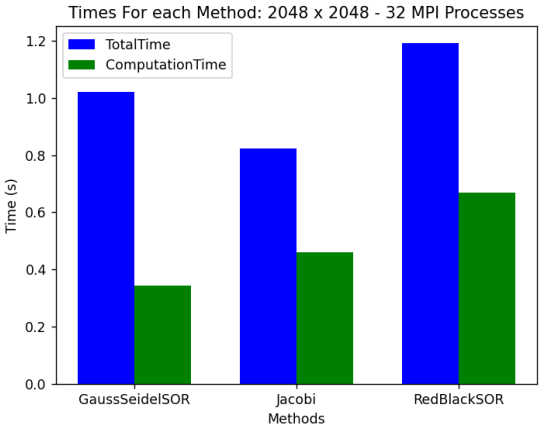
Στη συνέχεια παρουσιάζουμε διαγράμματα με αναλυτικότερη ανάλυση του συνολικού χρόνου, χρόνου επικοινωνίας και χρόνου υπολογισμού για να μελετήσουμε τις συμπεριφορές των μεθόδων και να επιβεβαιώσουμε τα προηγούμενα συμπεράσματα μας:



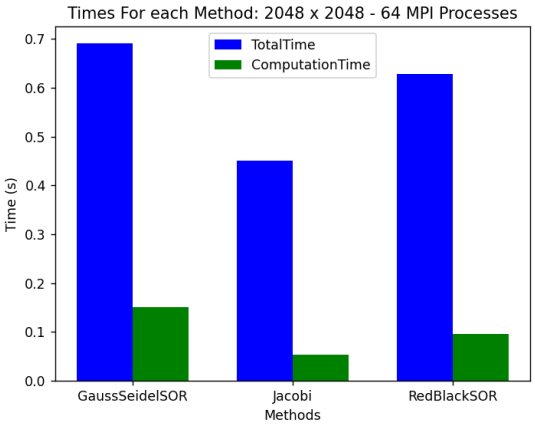
(α′) Times for Algorithms | Size 2048 x 2048 & 8 MPI Processes



(β′) Times for Algorithms | Size 2048 x 2048 & 16 MPI Processes

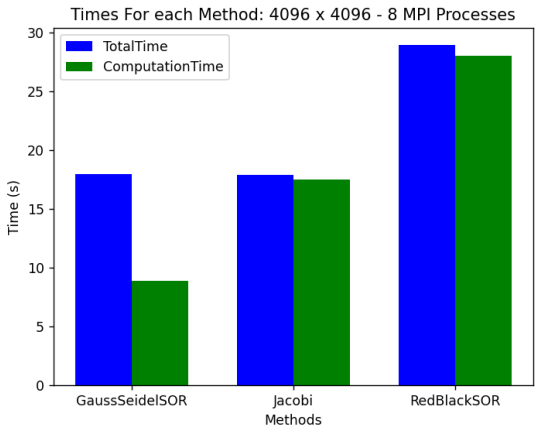


(γ′) Times for Algorithms | Size 2048 x 2048 & 32 MPI Processes

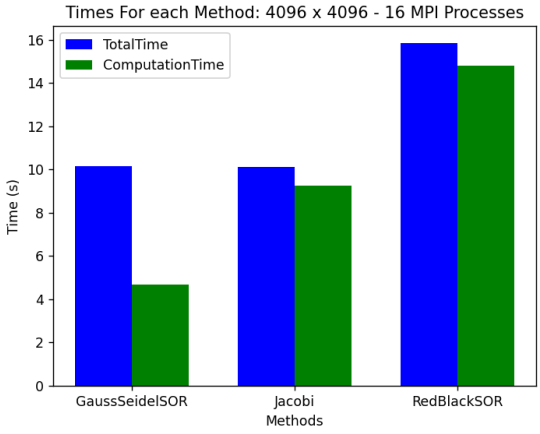


(δ′) Times for Algorithms | Size 2048 x 2048 & 64 MPI Processes

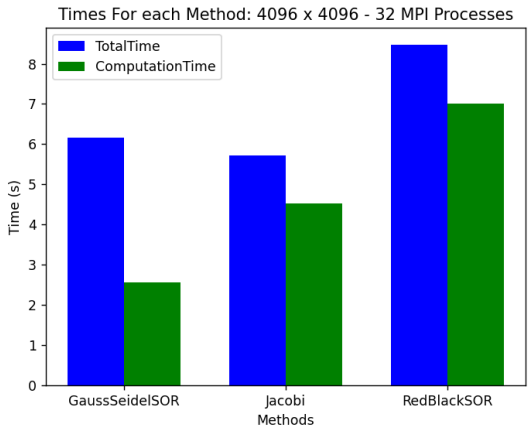
Σχήμα 1: Times for Algorithms for Size 2048 x 2048



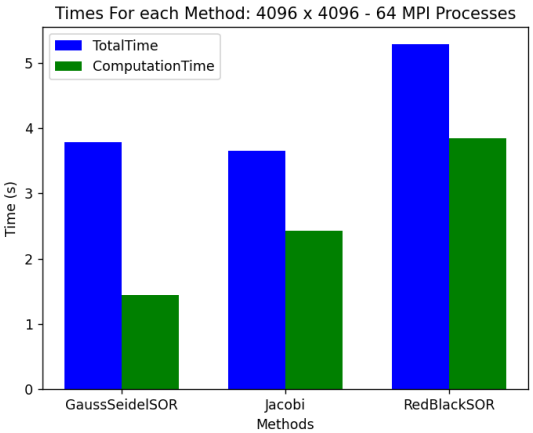
(α′) Times for Algorithms | Size 4096 x 4096 & 8 MPI Processes



(β′) Times for Algorithms | Size 4096 x 4096 & 16 MPI Processes

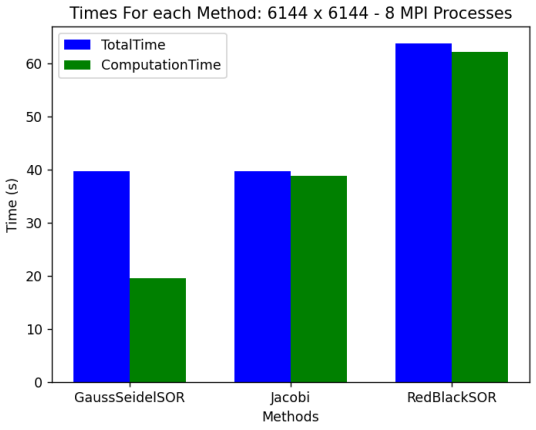


(γ′) Times for Algorithms | Size 4096 x 4096 & 32 MPI Processes

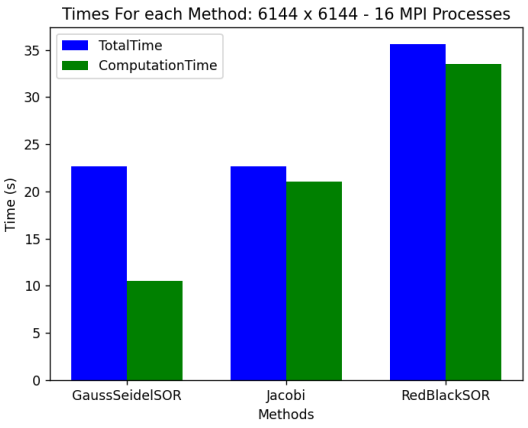


(δ′) Times for Algorithms | Size 4096 x 4096 & 8 MPI Processes

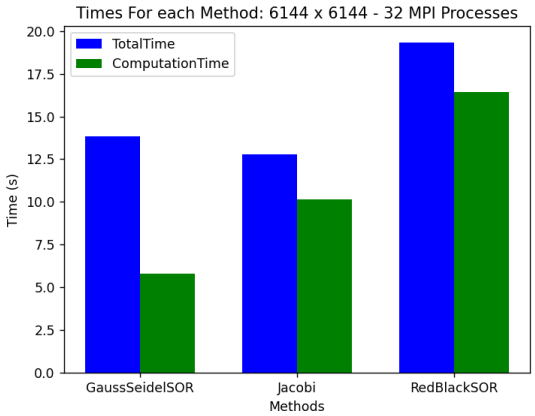
Σχήμα 2: Times for Algorithms for Size 4096 x 4096



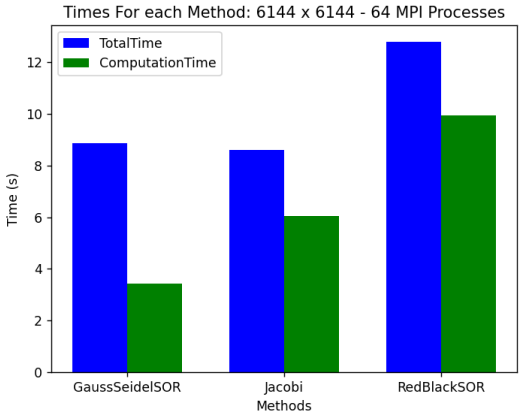
(α′) Times for Algorithms | Size 6144 x 6144 & 8 MPI Processes



(β′) Times for Algorithms | Size 6144 x 6144 & 16 MPI Processes



(γ′) Times for Algorithms | Size 6144 x 6144 & 32 MPI Processes



(δ′) Times for Algorithms | Size 6144 x 6144 & 64 MPI Processes

Σχήμα 3: Times for Algorithms for Size 6144 x 6144

Αν τώρα μελετήσουμε τους χρόνους για σταθερό μέγεθος iterations (T = 256), μπορούμε να δούμε πως ο Jacobi είναι ο πιο γρήγορος, λόγω της απλότητάς τους. Όμως αξίζει να προσέξουμε πως όσο μεγαλώνει ο αριθμός των processes τόσο μεγαλύτερο μέρος του χρόνου αφιερώνεται στο communication λόγω των blocking συναρτήσεων Send και Recv.

Το πιο σταθερό ration έχει ο GaussSeidelSOR, όπου ανεξαιρέτως του μεγέθους του πίνακα ή του αριθμού των διεργασιών, το computation time είναι το 1/2 με 1/3 του συνολικού χρόνου. Αυτό μας εξηγεί το καλό scaleup του αλγορίθμου, ο οποίος έχει ως μόνο μειονέκτημα το ότι δεν μπορεί να παραλληλοποιήσει εντελώς την εκτέλεσή του λόγω της εξάρτησης των στοιχείων του από τα αντίστοιχα πάνω και αριστερά στοιχεία.

Τέλος, ο RedBlackSOR, παρότι πετυχαίνει το καλύτερο speedup, έχει τους πιο αργούς χρόνους, λόγω της πολυπλοκότητάς του στον υπολογισμό των νέων πινάκων, αλλά και στην επικοινωνία μεταξύ των διεργασιών. Παράλληλα και εδώ έχουμε blocking συναρτήσεις Send και Recv με αποτέλεσμα να έχουμε μεγαλύτερο communication time για πολλά processes.

Παίρνοντας λοιπόν αυτά υπόψη, ως τελική απάντηση επιλέγουμε τον GaussSeidelSOR για την επίλυση του προβλήματος σε ένα σύστημα κατανεμημένης μνήμης, αν υποθέσουμε πως ασχολούμαστε με μεγάλες τιμές και με πολλές διεργασίες. Χάρη στο σταθερό του ratio και του καλού scale up από ένα σημείο και μετά ξεπερνά τον RedBlackSOR. Παράλληλα πετυχαίνει γρήγορη σύγκλιση όπως και ο RedBlackSOR και παρότι ο RedBlackSOR παραλληλοποιείται πλήρως, ο GaussSeidelSOR πετυχαίνει πολύ μικρότερα computation times.