Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Συστήματα Παράλληλης Επεξεργασίας

3η Εργαστηριακή Άσκηση

9ο Εξάμηνο - Ακαδημαϊκό Έτος 2020-2021

Γιαννόπουλος Εμμανουήλ - 03117031

Παγώνης Γεώργιος - 03117030

6 Ιανουαρίου 2021

1 Εισαγωγή

Σε αυτή την άσκηση σκοπός είναι να παραλληλοποιήσουμε τον υπολογισμό της εξίσωσης θερμότητας με 3 διαφορετικές μεθόδους (Jacobi, Gauss-Seidel SOR, Red-Black SOR) σε μοντέλο ανταλλαγής μηνυμάτων πάνω από κατανεμημένη αρχιτεκτονική. Λαμβάνουμε μετρήσεις τόσο για σταθερό αριθμό επαναλήψεων όσο και για έλεγχο σύγκλισης.

2 Υλοποίηση

Υλοποιήσαμε και τις 3 μεθόδους υπολογισμού με non-blocking επικοινωνία. Οι κώδικες (και κατ' επέκταση η στρατηγική παραλληλισμού με επαρκή σχόλια) βρίσκονται στο αρχείο που υποβλήθηκε.

3 Έλεγχος Σύγκλισης

Αρχικά παρουσιάζουμε τα αποτελέσματά μας για τον έλεγχο σύγκλισης. Οι μετρήσεις πραγματοποιούνται για μέγεθος πίνακα 1024×1024 και 64 MPI διεργασίες.

Μέθοδος	Total Time	Computation Time	Midpoint
Jacobi	516.186629	40.985089	5.431022
GaussSeidelSOR	13.042611	0.609010	5.642998
RedBlackSOR	2.446432	0.178110	5.642974

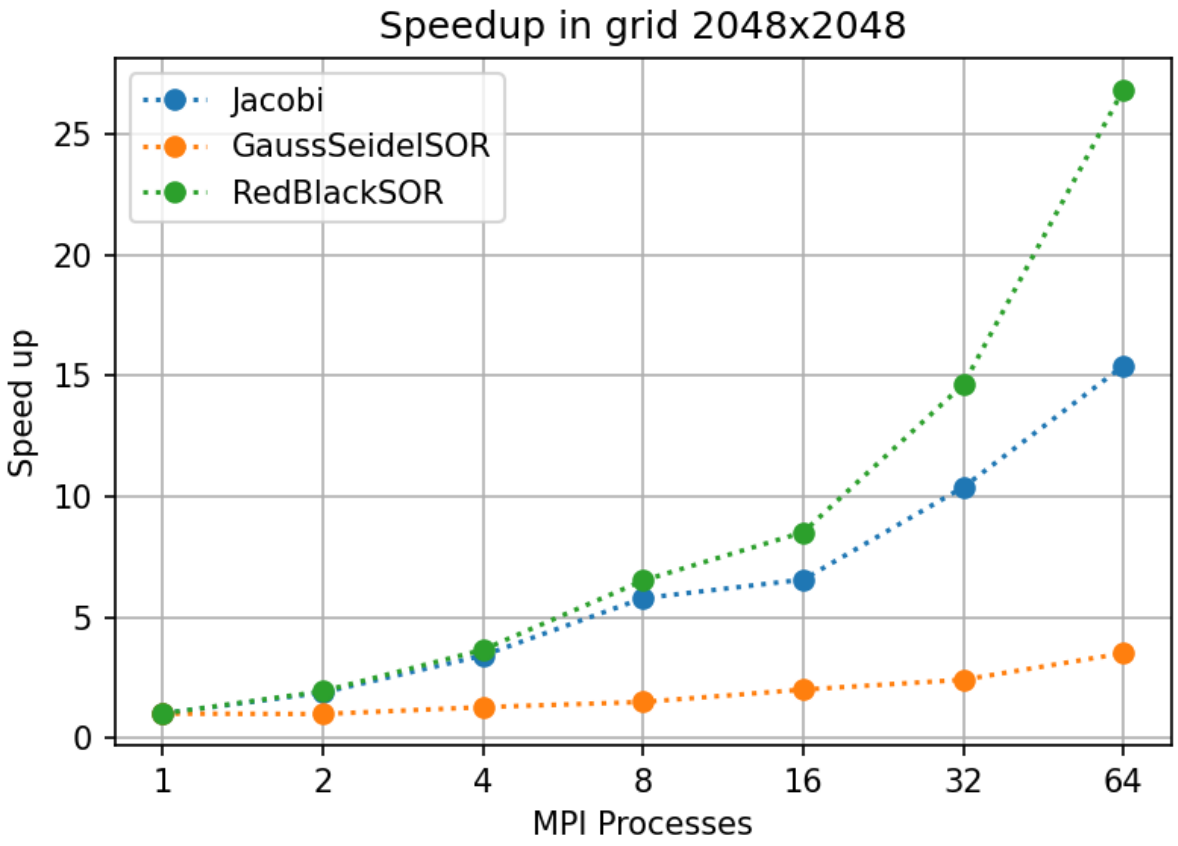
Παρατηρήσεις:

- Η μέθοδος Jacobi όπως περιμέναμε υστερεί κατά πολύ της μεθόδου Gauss-Seidel καθώς δεν αξιοποιεί τις ήδη υπολογισμένες τιμές της τωρινής χρονικής στιγμής.
- Η μέθοδος Red-Black φαίνεται να είναι μακράν η καλύτερη των τριών όντας περίπου 5 φορές πιο γρήγορη από την Gauss-Seidel και περίπου 200 φορές πιο γρήγορη από την Jacobi. Προφανώς θα επιλέγαμε αυτήν για την επίλυση του προβλήματος γιατί πετυχαίνει και το καλύτερο computation time και το καλύτερο total time με μεγάλη διαφορά.

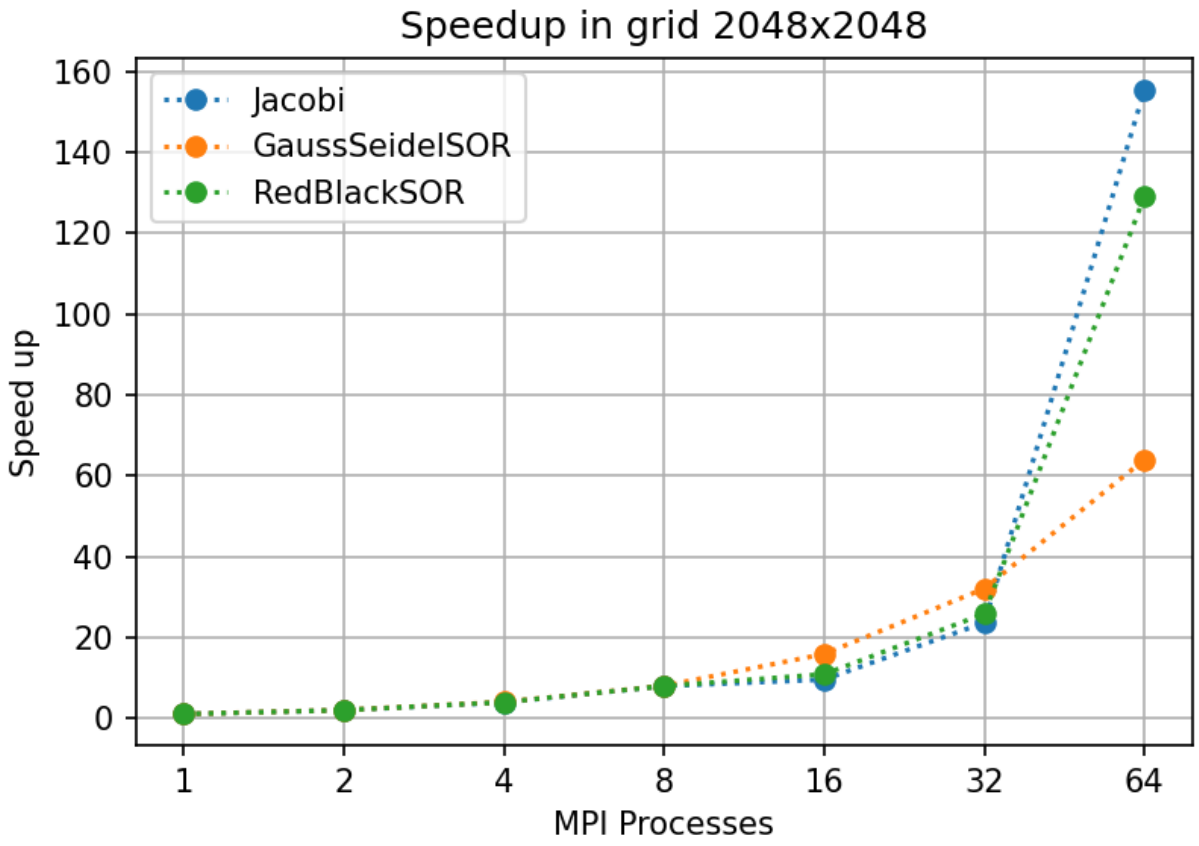
4 Σταθερός αριθμός επαναλήψεων

Πραγματοποιήσαμε επίσης μετρήσεις για σταθερό αριθμό επαναλήψεων της κάθε μεθόδου, σε μεγέθη πίνακα $2048 \times 2048, 4096 \times 4096, 6144 \times 6144$.

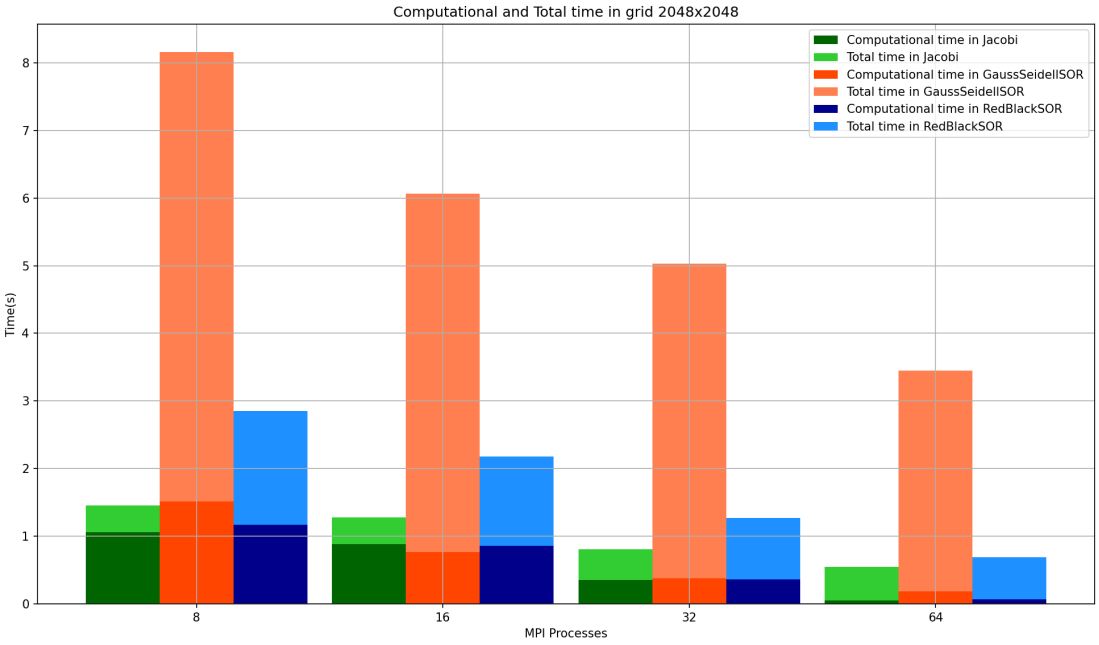
4.1 **Size** 2048×2048



Σχήμα 1: Total Time Speedup

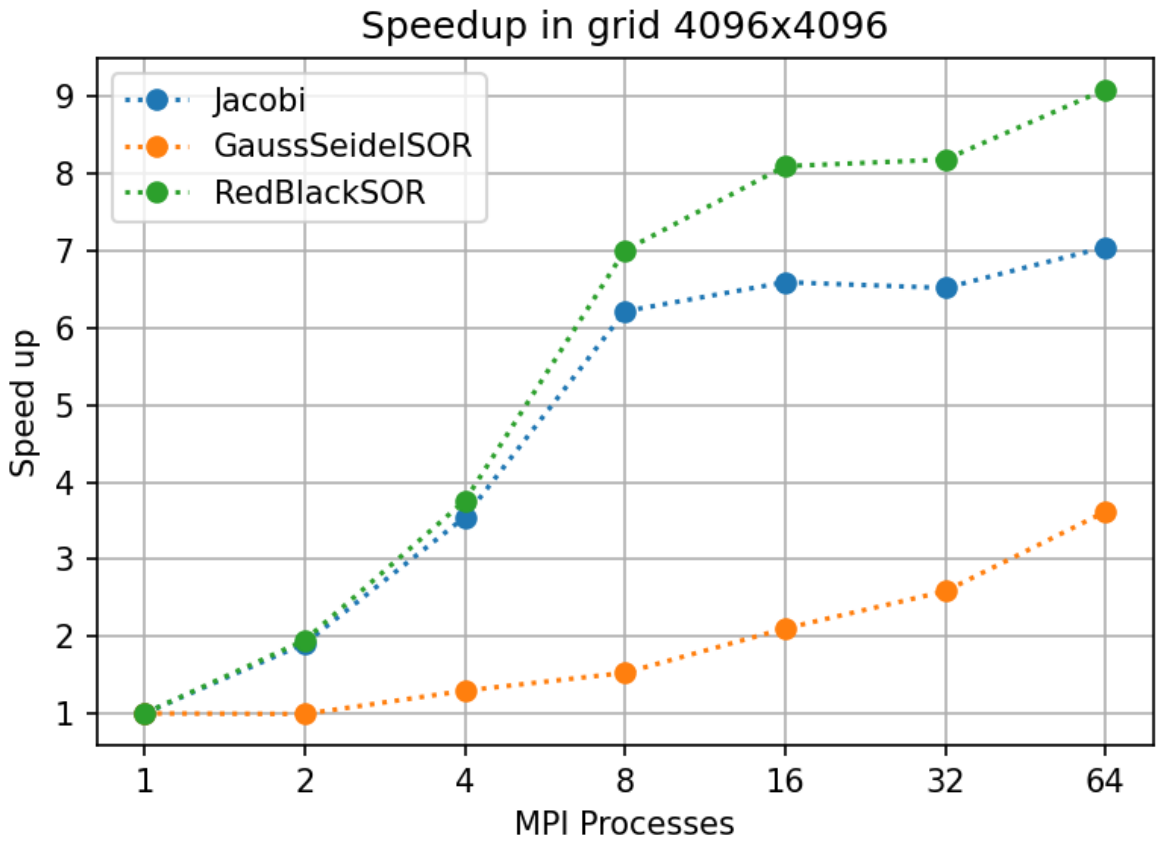


Σχήμα 2: Computation Time Speedup

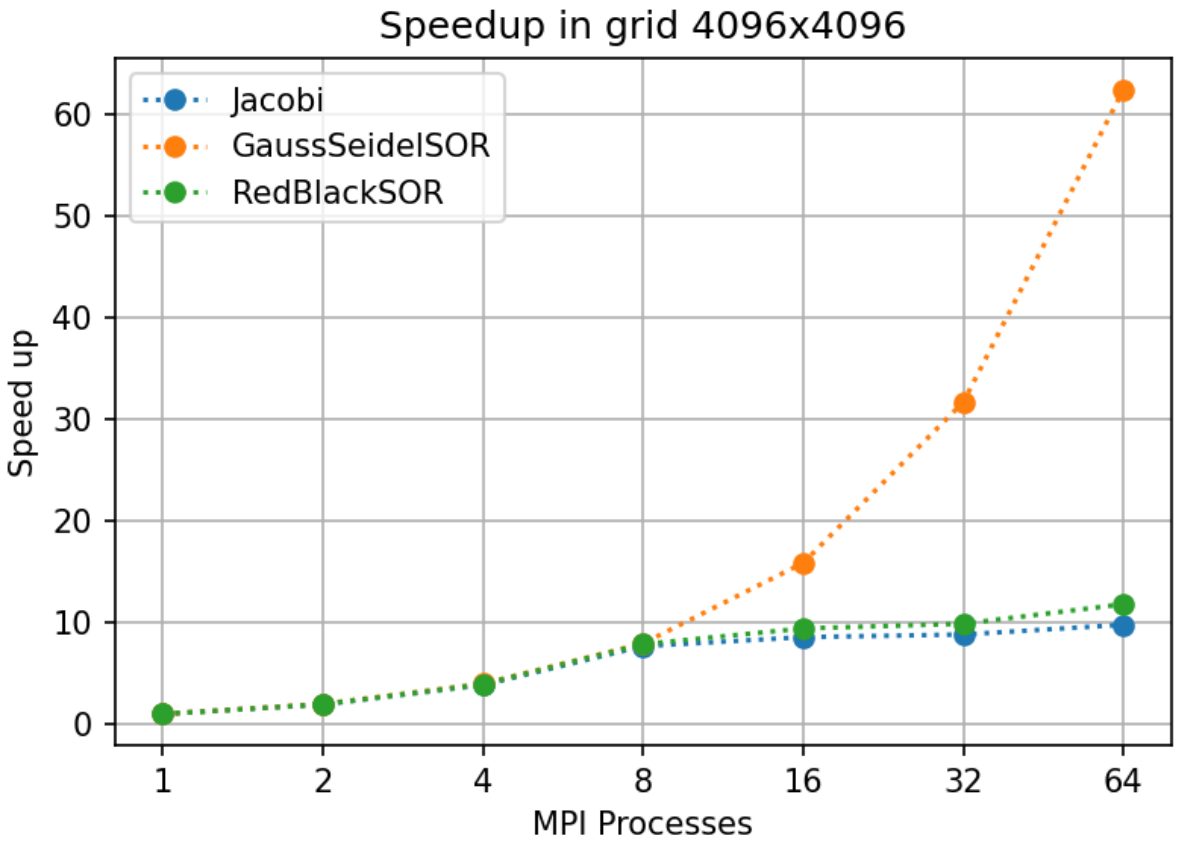


Σχήμα 3: Time Bars

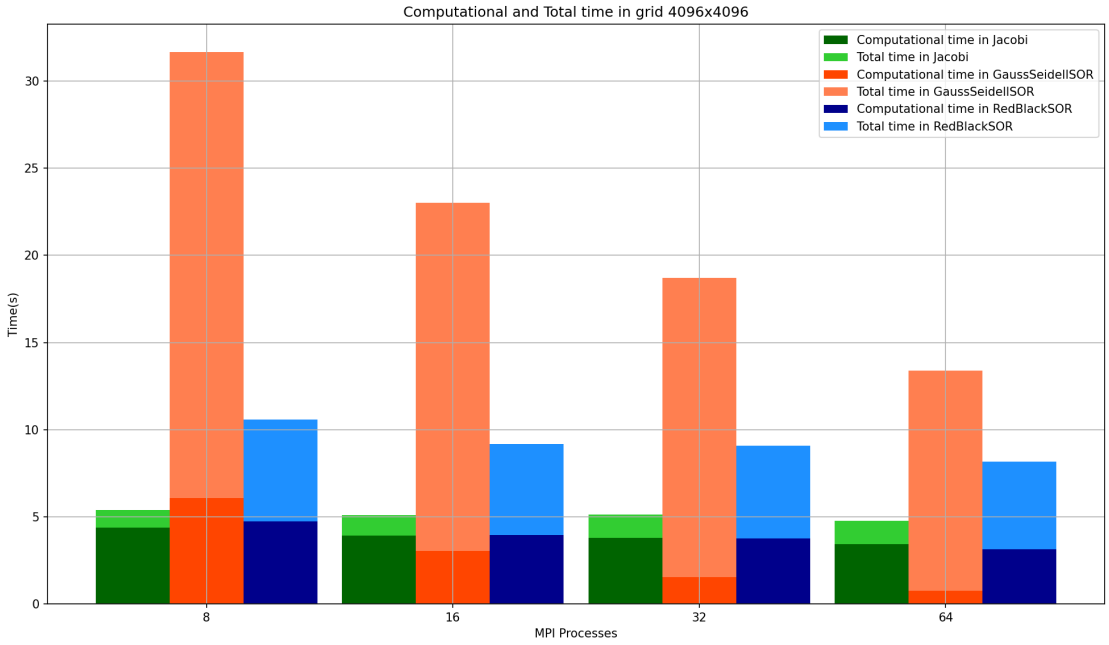
4.2 **Size** 4096×4096



Σχήμα 4: Total Time Speedup

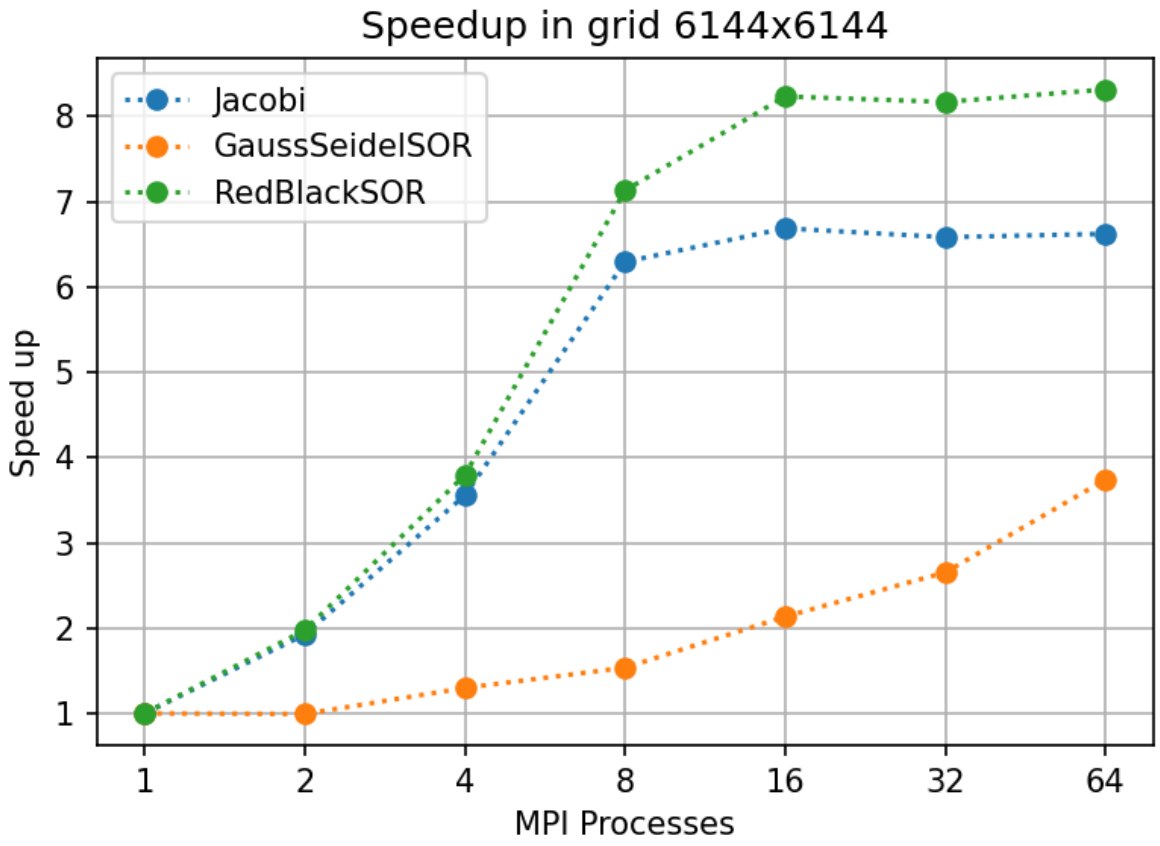


Σχήμα 5: Computation Time Speedup

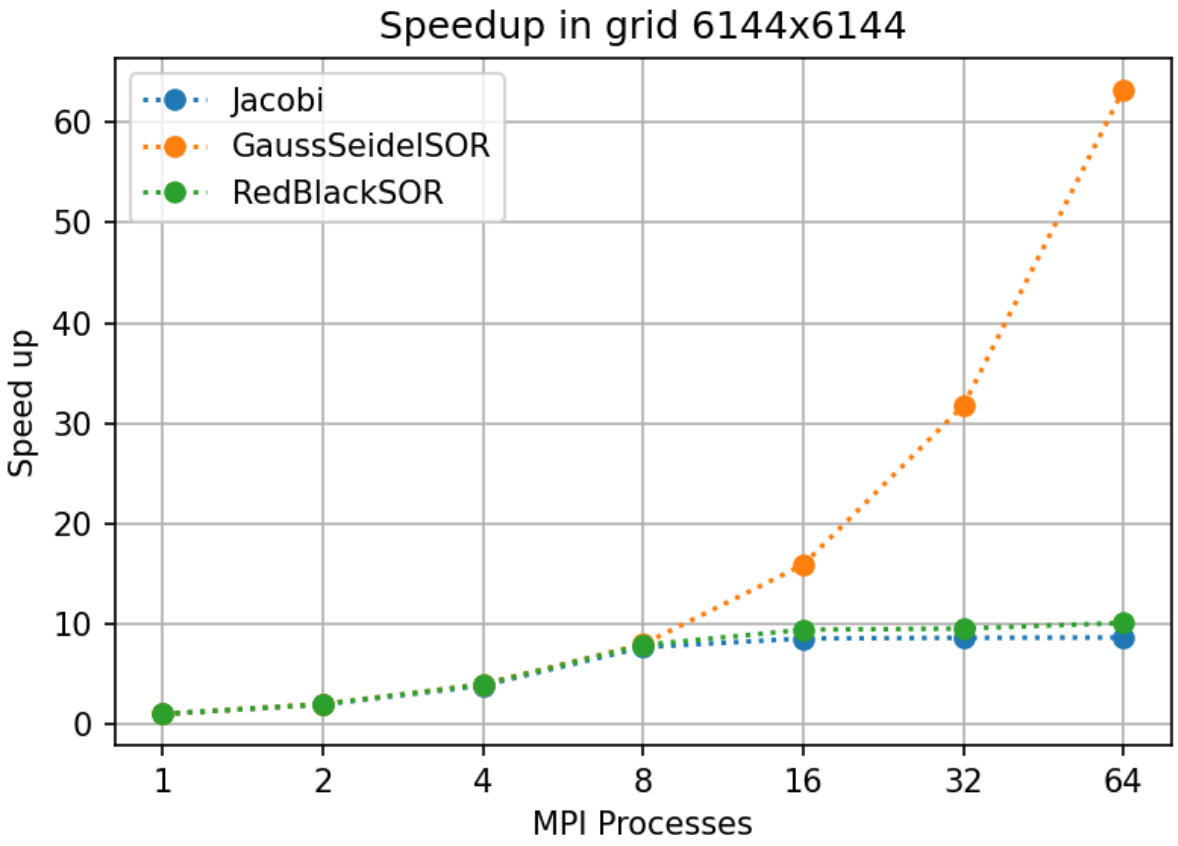


Σχήμα 6: Time Bars

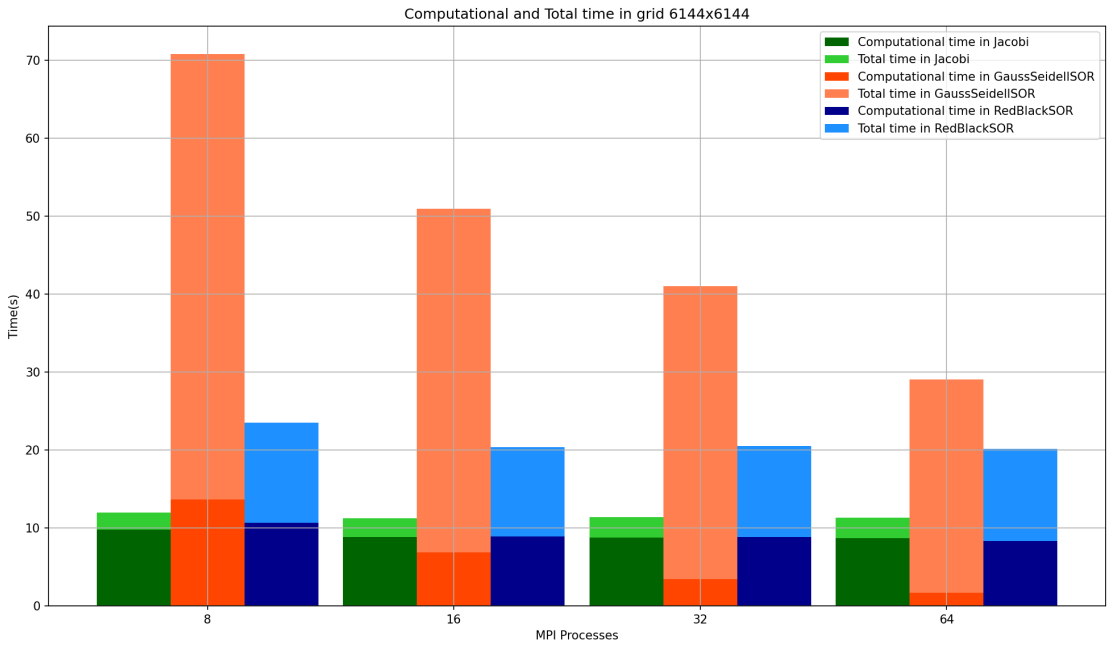
4.3 **Size** 6144×6144



Σχήμα 7: Total Time Speedup



Σχήμα 8: Computation Time Speedup



Σχήμα 9: Time Bars

Παρατηρήσεις:

- Όσον αφορά το συνολικό χρόνο εκτέλεσης, βλέπουμε ότι η μέθοδος Jacobi υπερτερεί των άλλων 2 μεθόδων κατά πολύ. Είδαμε πριν ότι η συγκεκριμένη μέθοδος αργεί πάρα πολύ σε συνθήκες σύγκλισης, αλλά για σταθερό χρόνο επαναλήψεων είναι η γρηγορότερη. Επίσης, η μέθοδος Gauss-Seidel-SOR που συνέκλινε πολύ γρηγορότερα από την Jacobi, τώρα έχει το χειρότερο συνολικό χρόνο εκτέλεσης.

- Όσον αφορά το χρόνο υπολογισμού, η μέθοδος Gauss-Seidel-SOR, αν και πολύ πιο αργή ως προς το συνολικό χρόνο εκτέλεσης, είναι η γρηγορότερη ως προς το χρόνο υπολογισμού για περισσότερους από 8 επεξεργαστές. Είναι επίσης η μέθοδος που βλέπουμε το καλύτερο speedup ως προς το χρόνο υπολογισμού.

- Όσον αφορά το speedup του συνολικού χρόνου υπολογισμού, όλες οι μέθοδοι έχουν πολύ καλύτερο scalability για τα μικρά μεγέθη πινάκων, πράγμα λογικό εφόσον για μεγαλύτερα μεγέθη θα χρειαστεί να αποστέλλονται και να λαμβάνονται περισσότερα στοιχεία. Η μέθοδος Red-Black SOR έχει το καλύτερο speedup από τις 3 μεθόδους, με τη μέθοδο Jacobi να ακολουθεί και την μέθοδο Gauss-Seidel να παρουσιάζει το χειρότερο.

- Όσον αφορά το speedup του computation time, η μέθοδος Gauss-Seidel παρουσιάζει σχεδόν linear speedup. Οι μέθοδοι Jacobi και Red-Black SOR κλιμακώνουν superlinearly για μέγεθος 2048×2048 και όχι πολύ καλά για τα υπόλοιπα μεγέθη. Για το τελευταίο μπορεί να οφείλεται σε αρκετούς παράγοντες όπως οι αυξημένες προσβάσεις στη μνήμη για μεγαλύτερα μεγέθη πίνακα, ή τα compiler optimizations που χρησιμοποιούμε.

- Οι διαφορές στα speedup, ιδιαίτερα του total time οφείλονται στα timings της επικοινωνίας μεταξύ των διεργασιών. Συγκεκριμένα στη μέθοδο Gauss-Seidel SOR οι διεργασίες πρέπει να

επικοινωνούν τόσο πριν όσο και μετά τους υπολογισμούς, πράγμα που τις αποσυγχρονίζει αρκετά και σπαταλάται περισσότερος χρόνος σε αναμονές. Αντίθετα, στη μέθοδο Jacobi υπάρχει μόνο ένα στάδιο ανταλλαγής στοιχείων και μετά γίνεται ο υπολογισμός. Αυτό ισχύει και για τη μέθοδο Red-Black SOR και γιαυτό οι 2 αυτές μέθοδοι παρουσιάζουν παρόμοιους χρόνους και speedups μόνο για το computation time.

5 Επεξήγηση κώδικα

5.1 Jacobi_mpi.c

Ο αλγόριθμος με τον οπόιο θα υπολογίσουμε το πρόβλημα είναι :

Algorithm 1 Gauss-Seidel-SOR

$$u_{x,y}^{t+1} = \frac{u_{x-1,y}^{t}+u_{x,y-1}^{t}+u_{x+1,y}^{t}+u_{x,y+1}^{t}}{4}$$

Για να μπορέσουμε να υπολογίσουμε τα στοιχεία που βρίσκονται στα σύνορα του local table που έχει η κάθε διεργασία του MPI πρέπει να έχουμε τα αντίστοιχα συνοριακά στοιχεία από τις διεργασίες **North, South, West, East**. Ακολουθούμε τα εξής βήματα για εφαρμόσουμε το MPI:

- Έχουμε 2 πίνακες έναν previous και έναν current, οπού για κάθε χρονική στιγμή χρησιμοποίουμε τον previous για να υπολογίσουμε τον current.
- Για να μπορεσούμε να μεταφέρουμε τα στοιχεία από τις γειτονικές διεργασίες προσθέτουμε row/col σε κάθε πλευρά.
- Κάνουμε Scatter ώστε κάθε διεργασία να έχει το δικό της κομμάτι από τον global και να τον αποθήκευσει στον δικό της local.
- Ορίζουμε ένα datatype column για να μπορούμε να μεταφέρουμε μια στήλη του πίνακα σε μία κλήση Send, Recv.
- Χρησιμοποίουμε την κλήση MPI_Cart_shift για βρούμε τις γείτονικες διεργασίες.
- Τροποποίουμε τα imin,imax,jmin,jmax κατάλληλα ανάλογα με την θέση που έχει η κάθε διεργασία στο grid.
- Κάνουμε async επικοινωνία ζητώντας και δίνοντας το συνοριακά στοιχεία του previous πίνακα στα γειτονικά στοιχεία με την χρήση MPI_Isend και MPI_Irecv. Κάθε επικοινωνία χαρακτηρίζεται από ένα MPI_Request. Για να κάνουμε συγχρονισμό κάνουμε MPI_Waitall για όλες τις κλήσεις περίμενοντας τις κλήσεις που έχουν τα δηλώμενα MPI_Request .
- Κάνουμε τον υπολογισμό. Αφού κάνουμε τον υπολογίσμο κάνουμε ένα MPI_Barrier για είμαστε σίγουροι ότι έχουν ολοκληρωθεί όλοι οι υπολογισμόι.
- Επαναλαμβάνουμε για όλες τις εποχές ή μέχρι να φτάσουμε σε σύγκλιση.
- Για να μπορούμε να τύπωσουμε τους χρόνους και το τελικό πίνακα κάνουμε MPI_Reduce τους τοπικούς χρόνους ttotal,tcomp,tconv στις global μεταβλητές total_time, comp_time,conv_time κρατώντας την μεγαλύτερη για να βρούμε τον μέγιστο χρόνο σε κάθε περίπτωση. Στην συνέχεια, εκτελούμε MPI_Gather για να "μαζέψουμε" τον τελικό πίνακα στην διεργασία με rank =0 που τυπώνει όλα τα αποτελέσματα.

- Για την σύγκλιση για να μπορούμε να δούμε αν συγκλίνει σε κάθε διεργασία ο δίκος της πίνακας, η κάθε διεργασία υπολογίζει το δικό της converged, κάνουμε ένα MPI_Allreduce στο global_converged κρατώντας την μικρότερη τιμή. Αν οι πίνακες κάθε διεργασίας συγκλίνουν τότε έχουμε φτάσει στην καθολίκη σύγκλιση.

5.2 GaussSeidelSOR_mpi.c

Πατάμε πάνω στον κώδικα του Jacobi. Η αλλαγή που κάνουμε είναι στο μοντέλο υπολογισμού.

Algorithm 2 Gauss-Seidel-SOR

$u_{x,y}^{t+1} = u_{x,y}^{t} + \omega * \frac{u_{x-1,y}^{t+1} + u_{x,y-1}^{t+1} + u_{x+1,y}^{t} + u_{x,y+1}^{t} - 4u_{x,y}^{t}}{4} \quad \omega \in (0,2)$

Τώρα χρείαζεται να αλλάξει και ο τρόπος επικοινωνίας. Το κάθε στοιχείο χρειάζεται να έχει το current των **North,West** και τα previous των **South,East**. Αντίστοιχα, πρέπει να στείλει το δικό του previous στα **North,West** και το δικό του current,αφού το υπολογίσει, στα **South,East**. Επομένως μια ενδεικτική λειτουργία είναι :

Algorithm 3 Mock up of Gauss-Seidel-SOR

for $t = 0$ to T **do**
    Receive from North this current
    Receive from West this current
    Receive from South this previous
    Receive from East this previous
    Send to North your previous
    Send to West your previous

    Calculate your current

    Send to South your current
    Send to East your current
end for

Οπότε ότι είναι πρίν από το στάδιο του Calculate μπορεί να γίνει παράλληλα , χρείαζεται να περίμενει το thread να γίνουν όλες οι επικοινωνίες, πράγμα που το υλοποίουμε με async communication, και στην συνέχεια, να στείλει παράλληλα το αποτέλεσμα του current στα γειτονικά που το χρείαζονται. Προφάνως, αυτή η υλοποίηση χάνει πολύ σε παραλληλισμό, γιατί για να προχωρήσει ένα thread στους υπολογισμούς του πρέπει να περιμένει να έχουν ολοκληρωθεί οι υπολογισμοί στα threads που είναι πάνω και δεξιά του. Ωστόσο, ότι το χάνει σε παραλληλισμό το κερδίζει στην σύγκλιση αφού συγκλίνει πολύ πιο γρήγορα από τον Jacobi.

5.3 RedblackSOR_mpi.c

Βασιζόμενοι και πάλι στον κώδικα του Jacobi τροποποίουμε τον κώδικα για τον τρόπο υπολογισμού:

Algorithm 4 Red-Black-SOR

if $(x+y) \mod 2 == 0$ **then**

$$u_{x,y}^{t+1} = u_{x,y}^{t} + \omega * \frac{u_{x-1,y}^{t} + u_{x,y-1}^{t} + u_{x+1,y}^{t} + u_{x,y+1}^{t} - 4u_{x,y}^{t}}{4} \quad \omega \in (0,2)$$

else if $(x+y) \mod 2 == 1$ **then**

$$u_{x,y}^{t+1} = u_{x,y}^{t} + \omega * \frac{u_{x-1,y}^{t+1} + u_{x,y-1}^{t+1} + u_{x+1,y}^{t+1} + u_{x,y+1}^{t+1} - 4u_{x,y}^{t+1}}{4} \quad \omega \in (0,2)$$

end if

Τώρα μπορούμε να το κάνουμε σε 2 φάσεις εφαρμόζοντας την ίδια λογική με την μέθοδο του Jacobi. Για να υπολογίσουμε τα Red στοιχεία πρέπει να έχουμε όλα τα δεδόμενα από την προήγούμενη χρόνική στιγμή των Black στοιχείων. Άρα κάνουμε επικοινωνία send-receive σε όλα τα γείτονικά **North, West, South, East** για τα previous στοιχεία τους. Υπολογίζουμε όλα τα Red στοιχεία. Τώρα που θέλουμε να πάρουμε από τα Red στοιχεία την current τιμή τους για να υπολογίσουμε τα Black, επικοινωνούμε με όλα τα γειτονικά για να τους στείλουμε τα current και υπολογίζουμε τα Black. Αυτή η μέθοδος συγκλίνει καλύτερα από την Gauss-Seidel και διωρθώνει τα προβλήματα που αντιμετωπίζουμε με την μεγάλη αναμονή για επικοινωνία που είχε από αυτή την μέθοδο.