Εργασία MPI – OpenMP – CUDA

1. Εισαγωγή

Υλοποιούμε ένα παράλληλο αλγόριθμο για το ατμοσφαιρικό μοντέλο. Ένα ατμοσφαιρικό μοντέλο προσομοιώνει ατμοσφαιρικές διεργασίες (αέρα, σύννεφα, υγρασία κτλ.) οι οποίες επηρρεάζουν το καιρό ή το κλίμα. Η προσομοίωση αυτή γίνεται με την επίλυση ενός συνόλου μερικών διαφορικών εξισώσεων. Η συμπεριφορά αυτών των εξισώσεων σε ένα συνεχές χώρο προσεγγίζεται απο την συμεριφορά τους σε ένα πεπερασμένο, διακριτό σύνολο ισοκατανεμημένων στο χώρο σημείων. Αυτά τα σημεία, στη γενικότερη περίπτωση, σχηματίζουν ένα ορθογώνιο πλέγμα μεγέθους $N_x \times N_y \times N_z$. Το πλέγμα είναι περιοδικό στις x και y διαστάσεις, δηλαδή το σημείο του πλέγματος $G_{0,0,0}$ θεωρείται γειτονικό του σημείου $G_{0,Ny-1,0}$.

Το ατμοσφαιρικό μοντέλο πραγματοποιεί μια χρονική ολοκήρωση για να προσδιορίσει τη κατάσταση της ατμόσφαιρας σε κάποια μελλοντική χρονική στιγμή. Η ολοκλήρωση προχωρά με μία σειρά διακριτών βημάτων, με κάθε βήμα να προχωρά τον υπολογισμό κατα μία σταθερή ποσότητα. Υποθέτουμε ότι το μοντέλο χρησιμοποιεί τη μέθοδο πεπερασμένης διαφοράς (finite difference) για την ενημέρωση των σημείων του πλέγματος, χρησιμοποιώντας ένα στένσιλ 9-σημείων για τον υπολογισμό της ατμοσφαιρικής κίνησης στο οριζόντιο επίπεδο, και ένα στένσιλ 3-σημείων στο κάθετο επίπεδο. Οι υπολογισμοί αυτοί αφορούν την επίλυση της μερικής διαφορικής εξίσωσης της δυναμικής των ρευστών για την κίνηση της ατμόσφαιρας.

Επιπλέον, το ατμοσφαιρικό μοντέλο εμπεριέχει τους υπολογισμούς φυσικής στο κάθετο επίπεδο οι οποίοι προσομοιώνουν την αλληλεπίδραση της ηλιακής ακτινοβολίας με την ατμόσφαιρα. Οι εξαρτήσεις δεδομένων μέσα στους υπολογισμούς φυσικής περιορίζονται μόνο στις κάθετες στήλες του πλέγματος.

Η εργασία χωρίζεται στα παρακάτω μέρη: α) ποιοτικός σχεδιασμός του αλγόριθμου με βάση τη μεθοδολογία Foster, αρχικά αφηρημένα και μετά κάνοντας τροποποιήσεις (όπως συγκερασμό διεργασιών), αν χρειάζεται, για τη βελτιστοποίηση της εκτέλεσης σε κάποια παράλληλη μηχανή, β) ανάλυση της απόδοσης του αλγορίθμου σε μια συγκεκριμένη παράλληλη μηχανή και δημιουργία ενός μοντέλου απόδοσης-κλιμάκωσης με κριτήριο ποιος συγκερασμός είναι καλύτερος, γ) Υλοποίηση του αλγορίθμου σύμφωνα με όσα προηγήθηκαν, δ) Μετρήσεις και μελέτη απόδοσης-κλιμάκωσης του προγράμματος και εξαγωγή συμπερασμάτων.

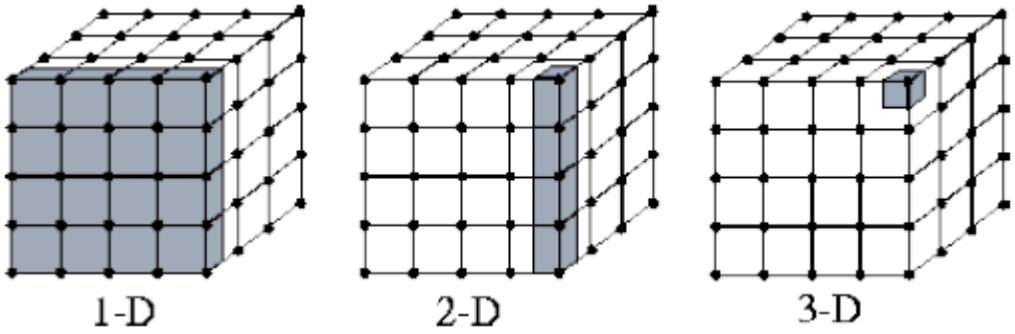
Ακολουθεί η υλοποίηση του ίδιου αλγόριθμου σε CUDA και σύγκριση των επιδόσεων με αυτές της υλοποίησης MPI.

2. Σχεδιασμός

Ο παράλληλος αλγόριθμος σχεδιάζεται ακολουθώντας τα 4 στάδια της μεθοδολογίας του Foster: partitioning, communication, agglomeration και mapping (PCAM).

2.1. Διαμερισμός Δεδομένων

Το ατμοσφαιρικό μοντέλο αναπαρίσταται με ένα τρισδιάστατο πλέγμα. Υπολογισμοί πραγματοποιούνται επαναλαμβανόμενα σε κάθε σημείο του πλέγματος. Είναι δυνατός ο διαμερισμός των δεδομένων σε 1, 2 ή 3 διαστάσεις:



Αρχικά επιλέγουμε το διαμερισμό σε 3 διαστάσεις (3-D) που είναι ο πιο «επιθετικός» και παρέχει τη μεγαλύτερη ελευθερία. Κάθε διεργασία περικλείει ένα σημείο του πλέγματος και είναι υπεύθυνη για τους υπολογισμούς σε αυτό.

Κατόπιν απαντούμε στα παρακάτω ερωτήματα (checklist) για να πάρουμε μια εικόνα του αν οι σχεδιαστικές επιλογές μας εμπεριέχουν προφανή λάθη ή όχι:

1. Ερ.: Ο διαμερισμός ορίζει τουλάχιστον μία τάξη μεγέθους παραπάνω διεργασίες από τους διαθέσιμους επεξεργαστές στο σύστημα-στόχο;
   Απ.: Ναι. Ο 3-D διαμερισμός ορίζει μέχρι και τόσες διεργασίες όσα είναι τα σημεία του πλέγματος, Στο σύστημα του εργαστηρίου έχουμε διαθέσιμους 40-50 επεξεργαστές, ενώ για ένα πλέγμα πχ. 256 x 256 x 64 έχουμε περίπου 4.000.000 διεργασίες, 4 τάξεις μεγέθους παραπάνω.

2. Ερ.: Ο διαμερισμός αποφεύγει περιττές απαιτήσεις υπολογισμών και αποθήκευσης;
   Απ.: Ναι. Οι υπολογισμοί για κάθε σημείο του πλέγματος γίνονται μία φορά σε κάθε επανάληψη και το αποτέλεσμα αποθηκεύεται επίσης μία φορά στην αντίστοιχη θέση.

3. Ερ.: Είναι οι διεργασίες συγκρίσιμου μεγέθους μεταξύ τους;
   Απ.: Ναι. Κάθε διεργασία έχει το ίδιο μέγεθος, ίσο με ένα σήμειο του πλέγματος.

4. Ερ.: Ο αριθμός των διεργασιών κλιμακώνει με το μέγεθος του προβλήματος;
   Απ.: Ναι. Ο αριθμός των διεργασιών κλιμακώνει γραμμικά με το μέγεθος του προβλήματος.

5. Ερ.: Υπάρχουν εναλλακτικές διαμερίσεις;
   Απ.: Ναι. Είναι δυνατές 1-D και 2-D διαμερίσεις

2.2. Επικοινωνία

Στο ατμοσφαιρικό μοντέλο, η επικοινωνία που χρειαζόμαστε μεταξύ των διεργασιών είναι κυρίως τοπική (local) και επίσης δομημένη (structured), στατική (static) και σύγχρονη (synchronous).

Τοπική, επειδή κάθε διεργασία επικοινωνεί με ένα σύνολο γειτονικών διεργασιών για τη πραγματοποίηση των υπολογισμών. Μπορεί να υπάρχει κι ενα μέρος καθολικής (global)

επικοινωνίας που αφορά το reduction που μπορεί να πραγματοποιείται ανα ορισμένα διαστήματα για έλεγχο.

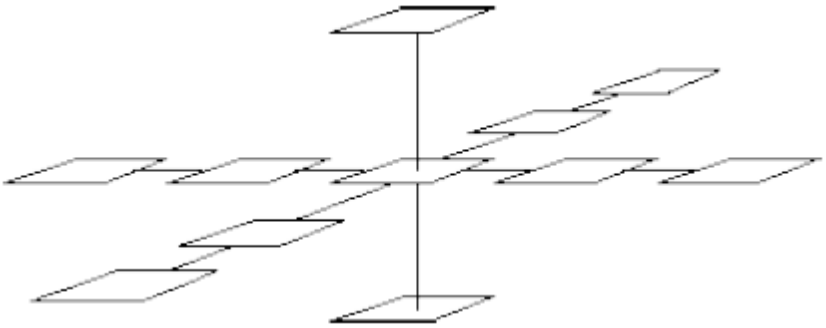
Δομημένη, επειδή η κάθε διεργασία και οι γειτονές της σχηματίζουν ένα πλέγμα με καλά καθορισμένα όρια.

Στατική, επειδή οι θέσεις των διεργασιών στο πλέγμα παραμένουν σταθερές σε όλη τη διάρκεια της εκτέλεσης. Δηλαδή κάθε διεργασία θα έχει πάντα τους ίδιους γείτονες τους οποίους χρειάζεται να εντοπίσει μόνο μία φορά, στην αρχή.

Σύγχρονη, επειδή κάθε διεργασία στέλνει τα δεδομένα της στις γειτονικές διεργασίες και και λαμβάνει απο αυτές. Έτσι κάθε διεργασία είναι ταυτόχρονα παραγωγός (producer) και καταναλωτής (consumer).

2.2.1. Τοπική Επικοινωνία

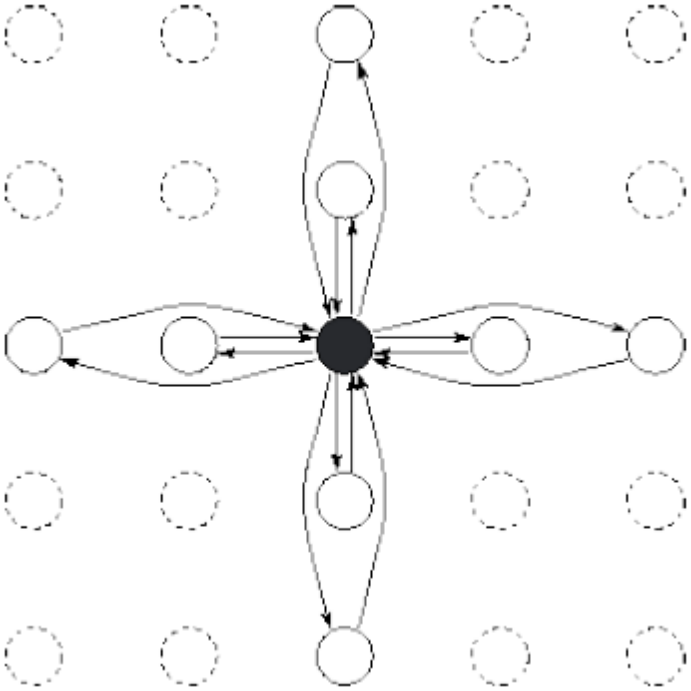
Οι υπολογισμοί που αφορούν το δυναμικό κομμάτι του ατμοσφαιρικού μοντέλου, αποτελούνται απο μια πεπερασμένη διαφορά Jacobi (Jacobi finite difference) σε ένα stencil 9 σημείων στην οριζόντια διάσταση (X, Y) και stencil 3 σημειών στη κάθετη διάσταση (Z).



Κάθε σημείο του πλέγματος ενημερώνεται, αντικαθιστώντας τη τιμή του με τη παρακάτω συνάρτηση των τιμών των γειτονικών στοιχείων στο stencil:

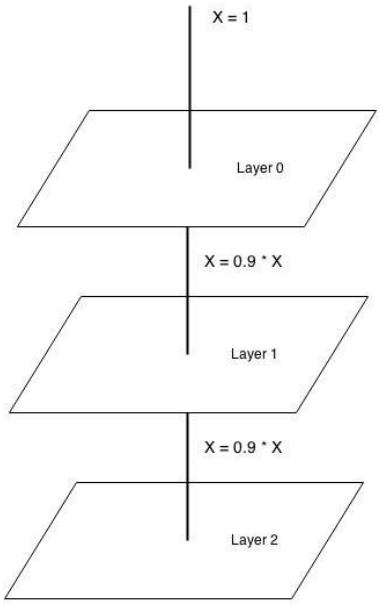
$$X_{i,j,k}^{(t+1)} = \frac{4X_{i,j,k}^{(t)} + X_{i-1,j,k}^{(t)} + X_{i-2,j,k}^{(t)} + X_{i+1,j,k}^{(t)} + X_{i+2,j,k}^{(t)} + X_{i,j-1,k}^{(t)} + X_{i,j-2,k}^{(t)} + X_{i,j+1,k}^{(t)} + X_{i,j+2,k}^{(t)} + X_{i,j,k-1}^{(t)} + X_{i,j,k+1}^{(t)}}{14}$$

Αν υποθέσουμε ότι δημιουργείται μία διεργασία ανά σημείο του πλέγματος, τότε για να υπολογίσει μια διεργασία το δικό της σημείο, πρέπει να γνωρίζει τα σημεία που έχουν υπολογίσει οι γειτονικές της διεργασίες στο stencil. Ορίζουμε έτσι 2 κανάλια επικοινωνίας μεταξύ κάθε ζεύγους διεργασιών, ένα για αποστολή και ένα για λήψη. Κάθε διεργασία: α) στέλνει το $X_{i,j,k}^{(t)}$ σε κάθε γείτονα,
β) λαμβάνει τα $X_{i-1,j,k}^{(t)}$ $X_{i-2,j,k}^{(t)}$ $X_{i+1,j,k}^{(t)}$ $X_{i+2,j,k}^{(t)}$ $X_{i,j-1,k}^{(t)}$ $X_{i,j-2,k}^{(t)}$ $X_{i,j+1,k}^{(t)}$ $X_{i,j+2,k}^{(t)}$ $X_{i,j,k-1}^{(t)}$ $X_{i,j,k+1}^{(t)}$ απο τους γείτονες, γ) υπολογίζει το $X_{i,j,k}^{(t+1)}$. Έτσι, όσον αφορά το stencil 9-σημείων στο οριζόντιο επίπεδο, η επικοινωνία θα έχει ως εξής:



Και αντίστοιχα για το stencil 3-σημείων στο κάθετο επίπεδο. Συνολικά απαιτούνται (2 × 8) + (2 × 2) = 20 μηνύματα ανά διεργασία και ανά βήμα για τον υπολογισμό του Jacobi.

Στη κάθετη διεύθυνση υπολογίζεται η συνάρτηση z(X) = pow(X,10) που αναπαριστά την ένταση της ηλιακής ακτινοβολίας X σε κάποιο ύψος z. Κάθε «στρώμα» της ατμόσφαιρας απορροφά ένα 10% της ακτινοβολίας. Για τον υπολογισμό της συνάρτησης στο κάθετο επίπεδο, ξεκινώντας απο την κορυφή του πλέγματος, κάθε διεργασία σε αυτό θα πρέπει να λαμβάνει τη τιμή του X απο την απο πάνω της διεργασία και να στέλνει τη νέα τιμή που υπολόγισε στην απο κάτω της διεργασία:



Το κομμάτι το υπολογισμού της συνάρτησης παρουσιάζει δυσκολίες τόσο απο πλευράς απόδοσης όσο και υλοποίησης, καθώς η επικοινωνία δεν μπορεί να προχωρήσει παράλληλα αφού ο υπολογισμός σε κάθε διεργασία είναι εξαρτώμενος απο τη τιμή που υπολογίστηκε απο τη

προηγούμενη απο αυτή διεργασία. Επίσης θα πρέπει να τροποποιηθεί ο σειριακός κώδικας για τον υπολογισμό της συνάρτησης. Θα δούμε παρακάτω ότι οι δυσκολίες αυτές μπορούν να αποφευχθούν με τον συγκερασμό.

2.2.2. Καθολική Επικοινωνία

Το ατμοσφαιρικό μοντέλο υπολογίζει περιοδικά την ολική μάζα της ατμόσφαιρας, η οποία είναι κατανεμημένη στις διεργασίες, για να διαπιστώσει ότι η προσομοίωση προχωρά χωρίς λάθη. Αυτό απαιτεί την εκτέλεση ενός parallel reduction με τελεστή πρόσθεσης. Η λειτουργία αυτή θα προσθέτει σε μία τιμή τις $N_x \times N_y \times N_z$ τιμές απο τις $N_x \times N_y \times N_z$ διεργασίες:

$$\text{Total Mass} = \sum_{i=0}^{N_{x-1}} \sum_{j=0}^{N_{y-1}} \sum_{k=0}^{N_{z-1}} M_{ijk}$$

Αξιολογούμε το σχεδιασμό της επικοινωνίας για τον παράλληλο αλγόριθμό μας με το παρακάτω checklist:

1. Ερ.: Πραγματοποιούν όλες οι διεργασίες τον ίδιο αριθμό επικοινωνιακών λειτουργιών;
   Απ.: Ναι. Κάθε διεργασία επικοινωνεί με άλλες 10 (8+2) στο stencil, στέλνωντας και λαμβάνοντας.

2. Ερ.: Επικοινωνεί κάθε διεργασία μόνο με ένα μικρό αριθμό γειτόνων;
   Απ.: Ως επι το πλείστον ναι. Κάθε διεργασία επικοινωνεί με τους γειτόνους τις στο stencil, αλλά ανά τακτά διαστήματα πραγματοποιείται καθολική επικοινωνία μέ το reduction, όπου όλες οι διεργασίες αθροίζουν τη μάζα τους σε μία.

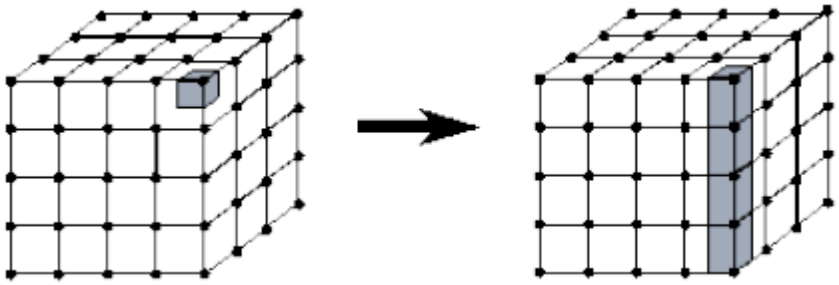
3. Ερ.: Οι επικοινωνιακές λειτουργίες πραγματοποιούνται παράλληλα;
   Απ.: Όχι όλες. Οι επικοινωνία που αφορά το stencil πραγματοποιείται παράλληλα αλλά όχι η επικοινωνία που αφορά τον υπολογισμό της συνάρτησης και αυτό θα οδηγήσει σε μη βέλτιστη απόδοση του αλγορίθμου. Το πρόβλημα αυτό μπορεί να ξεπεραστεί με το συγκερασμό.

4. Ερ.: Οι υπολογισμοί που σχετίζονται με διαφορετικές διεργασίες μπορούν να πραγματοποιούνται παράλληλα;
   Απ.: Οι υπολογισμοί που αφορούν το Jacobi πραγματοποιούνται παράλληλα, αλλά οι υπολογισμοί που αφορούν τη συνάρτηση στο κάθετο επίπεδο πραγματοποιούνται υποχρεωτικά σειριακά, κάτι που θα έχει αρνητικό αντίκτυπο στην απόδοση. Το πρόβλημα αυτό μπορεί να ξεπεραστεί με το συγκερασμό.

2.3. Συγκερασμός - Αντιστοίχηση

Σε αυτό το στάδιο επανεξετάζουμε πρακτικά το διαμερισμό που αφηρημένα είχαμε κάνει στο 1^{ο} στάδιο, με σκοπό να κάνουμε τον αλγόριθμο πιο αποδοτικό όταν θα εκτελεστεί σε κάποια παράλληλη μηχανή.

Το σημείο που επαναξετάζουμε αρχικά είναι ο διαμερισμός στο κάθετο επίπεδο. Όπως αναφέραμε, στο κάθετο επίπεδο απαιτείται επικοινωνία όχι μόνο για το stencil 3-σημείων αλλά και για τον υπολογισμό της συνάρτησης που είναι δύσκολο να εκτελεστεί παράλληλα. Παρατηρώντας ότι οι υπολογισμοί περιορίζονται μόνο στις κάθετες στήλες, μπορούμε να συγκεράσουμε σε μία όλες τις διεργασίες μέσα σε κάθε κάθετη στήλη.



Η απόφαση αυτή μας δίνει 2 πλεονεκτήματα:

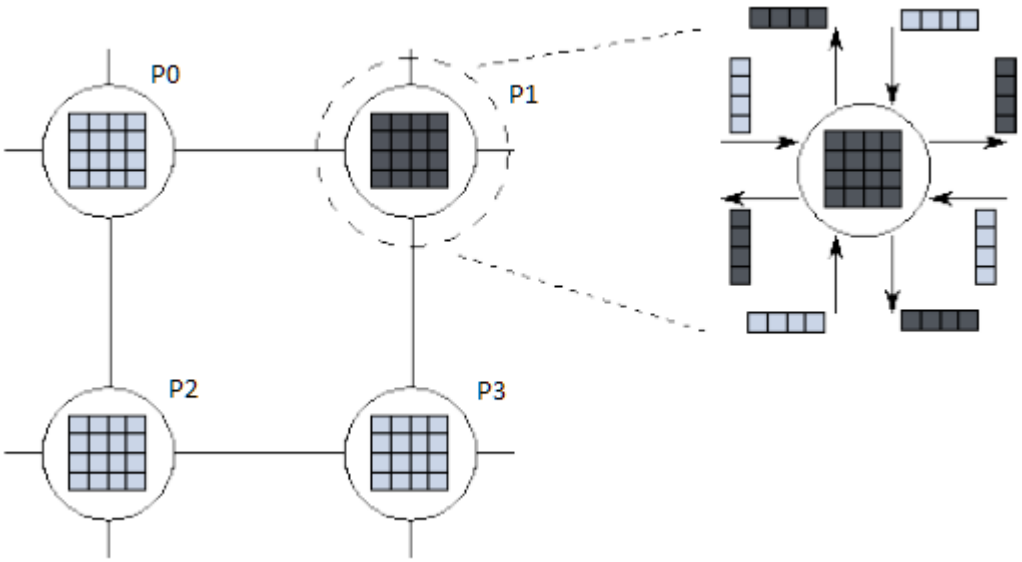
α) Η απαιτούμενη επικοινωνία μειώνεται, τόσο για το stencil από 20 μηνύματα ανα διεργασία και ανά βήμα σε 16 μυνήματα ανα διεργασία και ανα βήμα, όσο και για τον υπολογισμό της συνάρτησης που τώρα δεν απαιτεί καθόλου επικοινωνία. Έτσι αναμένουμε καλύτερη απόδοση και κλιμάκωση του αλγορίθμου.

β) Η υλοποίηση απο προγραμματιστικής πλευράς γίνεται ευκολότερη, καθώς ο ακολουθιακός κώδικας για τον υπολογισμό της συνάρτησης στο κάθετο επίπεδο μπορεί να ενσωματωθεί στο παράλληλο πρόγραμμα χωρίς αλλαγές.

Έτσι απο τον 3-D διαμερισμό που είχαμε κάνει θεωρητικά αρχικά, στη πράξη συμφέρει να πέσουμε σε 2-D καθώς αυτός ο διαμερισμός αναμένουμε να είναι ο βέλτιστος τόσο απο πλευράς απόδοσης, όταν θα εκτελέσουμε τον αλγόριθμο σε κάποια παράλληλη μηχανή όσο και απο πλευράς ευκολίας υλοποίησης.

Περνώντας στο συγκερασμό στο οριζόντιο επίπεδο, αποφασίζουμε κάθε διεργασία να περικλείει ένα μέρος του πλέγματος και να δημιουργούνται τόσες διεργασίες όσοι είναι οι επεξεργαστές στο σύστημα που θα εκτελεστεί ο αλγόριθμος, φτιάχνοντας ένα SPMD πρόγραμμα. Δηλαδή αν υπάρχουν P επεξεργαστές στη παράλληλη μηχανή, θα δημιουργηθούν P διεργασίες και καθεμία θα περικλείει ένα $\frac{N_x}{\sqrt{P}} \times \frac{N_y}{\sqrt{P}} \times N_z$ κομμάτι του πλέγματος. Κάθε διεργασία στέλνει και λαμβάνει τις συνοριακές γραμμές και στήλες (halo points) προς και από τις γειτονικές διεργασίες που βρίσκονται σε γειτονικούς επεξεργαστές.

Αυτή η κατανομή επιλύει και το θέμα της αντιστοίχησης διεργασιών σε επεξεργαστές (mapping), οπότε έχουμε σε μεγάλο βαθμό τελείωσει με το στάδιο του σχεδιασμού.



Αξιολογώντας με βάση το checklist του Foster, έχουμε:

1. Ερ.: Ο συγκερασμός έχει περιορίσει το κόστος επικοινωνίας αυξάνοντας τη τοπικότητα;
   Απ.: Ναι, με τον ολικό συγκερασμό της $3^{ης}$ διάστασης αναιρέθηκε η ανάγκη για επικοινωνία για τον υπολογισμό της συνάρτησης και στο οριζόντιο επίπεδο κάθε διεργασία περιλαμβάνει περισσότερα απο ένα σημεία του πλέγματος ανάλογα με το συνολικό μέγεθος του πλέγματος και τον αριθμό των επεξεργαστών

2. Ερ.: Ο συγκερασμός έχει ως αποτέλεσμα διεργασίες με παρόμοια κόστη επικοινωνίας και υπολογισμών;
   Απ.: Ναι, δημιουργείται 1 διεργασία ανα επεξεργαστή και το πλέγμα μοιράζεται ισόποσα στις διεργασίες.

3. Ερ.: Ο αριθμός των διεργασιών κλιμακώνει με το μέγεθος του προβλήματος;
   Απ.: Ναι, ο αριθμός των διεργασιών κλιμακώνει γραμμικά με τον αριθμό των επεξεργαστών, εφόσον δημιουργείται 1 διεργασία ανα επεξεργαστή, και με το μέγεθος του προβλήματος. Έτσι θα μπορεί να επιλύσει και μεγαλύτερα προβλήματα σε συστήματα με περισσότερο επεξεργαστές διατηρώντας καλή απόδοση.

4. Ερ.: Αν ο συγκερασμός μείωσε τις ευκαιριές για παράλληλη εκτέλεση, έχει απομείνει αρκετή παραλληλία για τις σημερινές και μελλοντικές παράλληλες μηχανές;
   Απ.: Ναι. Ο συγκερασμός της $3^{ης}$ διάστασης αφαίρεσε την παραλληλία απο αυτή τη διάσταση, η οποία όμως ερχόταν με μεγάλα κόστη επικοινωνίας. Στο οριζόντιο επίπεδο υπάρχει η παραλληλία είναι αρκετή ώστε να τροφοδοτήσει μεγάλο αριθμό επεξεργαστών.

5. Ερ.: Μπορεί ο αριθμός των διεργασιών να μειωθεί περαιτέρω χωρίς να προκληθούν προβλήματα διαμοιρασμού του φόρτου, πολυπλοκότητας υλοποίησης ή μειωμένης κλιμάκωσης;
   Απ.: Εφόσον δημιουργείται ακριβώς μία διεργασία ανα επεξεργαστή, ο αριθμός των διεργασιών δε μπορεί να μειωθεί περισσότερο. Το πλέγμα του μοντέλου κατανέμεται ισόποσα στις διεργασίες – επεξεργαστές. Έτσι έχουμε εναν αλγόριθμο που δημιουργεί το μικρότερο δυνατό αριθμό διεργασίων μεγάλης κοκκότητας (large-grained tasks).

6. Ερ.: Ποιά είναι τα κόστη που απαιτούνται για την τροποποίηση ήδη υπάρχοντος ακολουθιακού κώδικα ώστε να εκτελείται παράλληλα;
   Απ.: Στο οριζόντιο επίπεδο, ο αλγόριθμος του Jacobi είναι τετριμμένα παραλληλοποιήσιμος (trivially parallelizable). Στο κάθετο επίπεδο, το κόστος παραλληλοποίησης του υπολογισμού της συνάρτησης της ηλιακής ακτινοβολίας είναι μεγάλο αλλά μέσω του συγκερασμού της $3^{ης}$ διάστασης μπορούμε να ενσωματώσουμε χωρίς αλλαγές τον ακουλουθιακό κώδικα για τον υπολογισμό της.

3. Σχεδιασμός με βάση ποσοτικά δεδομένα

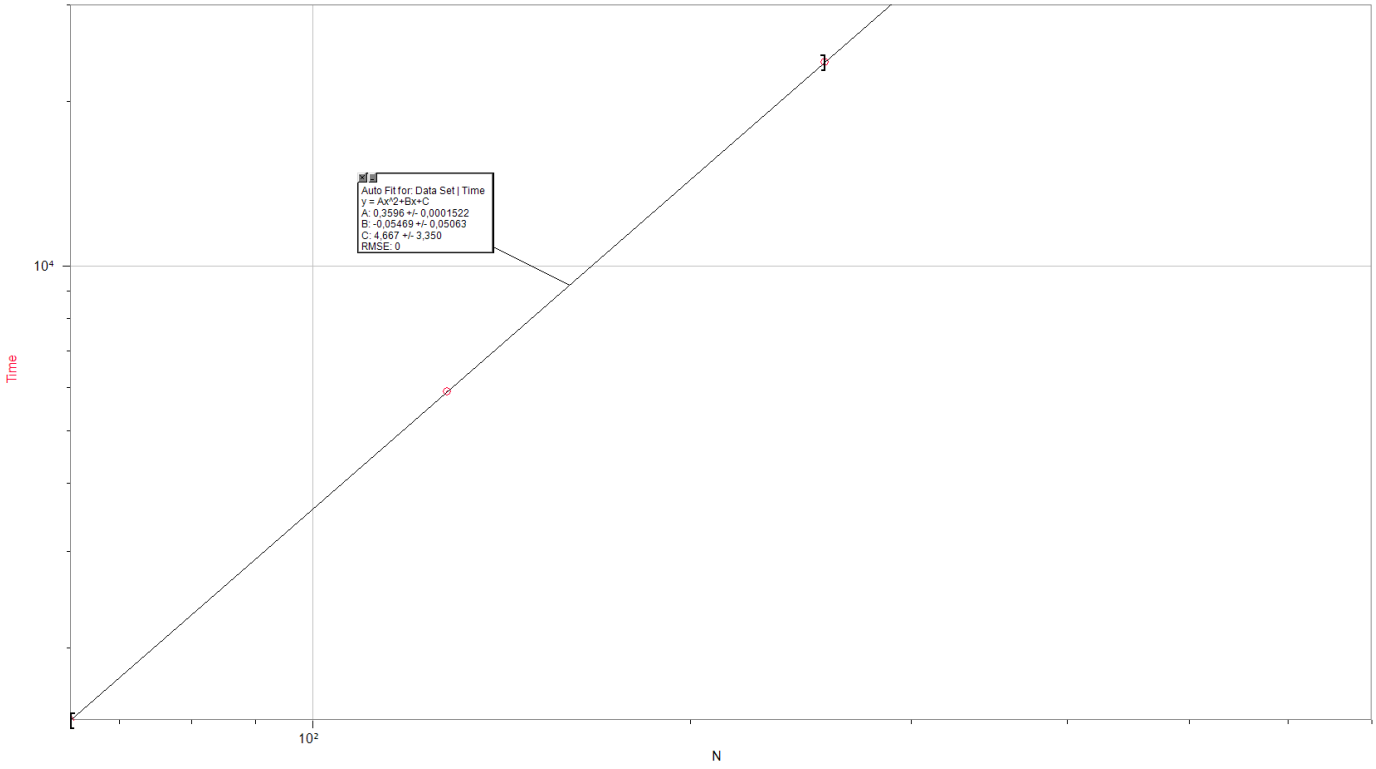
Σε αυτό το στάδιο θα αναλύσουμε την απόδοση σε ποσοτική βάση προκειμένου να θεμελιώσουμε μαθηματικά την επιλογή του 2-D συγκερασμού που επιλέξαμε πριν.

Έστω ότι t_c είναι ο χρόνος που απαιτείται για τον υπολογισμό ενός σημείου ενός N × N × Z πλέγματος. Τότε ο συνολικός χρόνος που απαιτείται τους υπολογισμούς είναι: $T_{comp} = t_c N^2 Z$.

Για την εύρεση του t_c, μετράμε το χρόνο T_{comp} που χρειάζεται για να εκτελεστεί το αντίστοιχο ακολουθιακό πρόγραμμα που εκτελεί τους ίδιους υπολογισμούς με το παράλληλο. Οι χρόνοι εκτέλεσης (msec) για διάφορες τιμές του N (κρατάμε το Z = 16 σταθερό) και 200 επαναλήψεις καταχωρούνται στον παρακάτω πίνακα, κάθε μέτρηση επαναλάμβανεται 3 φορές και παίρνουμε το μέσο όρο των μετρήσεων:

N	T_1	T_2	T_3	T_{comp}
64	1472	1473	1476	1474
128	5888	5895	5884	5889
256	23558	23550	23559	23556
512	94314	94558	94026	94299

Παρατηρούμε ότι η διαφοροποίηση στους χρόνους εκτέλεσης του ακολουθιακού προγράμματος είναι πολύ μικρή και ο χρόνος αυξάνεται γραμμικά με το N^2 όπως αναμενόταν. Οι άξονες είναι σε λογαριθμική κλίμακα:

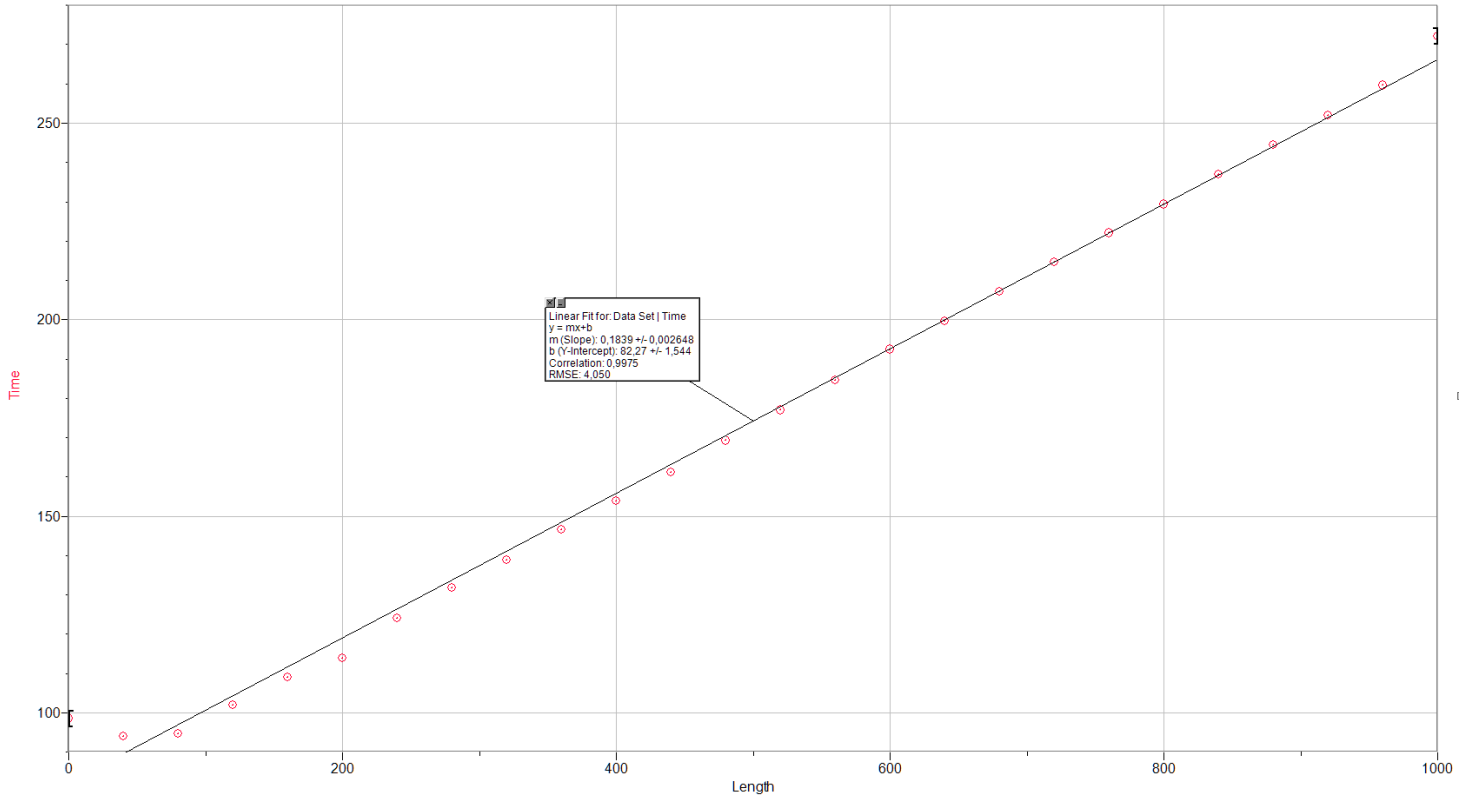


Απο το fitting των μετρήσεων με τη συναρτηση t_cN^2Z προκύπτει ότι:

$$t_cZ = 0.3596\ msec \rightarrow t_c = \frac{0.3596}{16} = 0.022475\ msec = 22.5\ \mu s$$

Τέλος, τα κόστη εκκίνησης (startup cost, t_s) και μεταφοράς μυνήματος (transfer cost, t_w) για το δίκτυο Ethernet του εργαστηρίου βρέθηκαν τρέχοντας το mpptest σε halo exchange mode, με μία διεργασία ανα μηχανή. Έτσι μετράμε το κόστος της επικοινωνίας για την ανταλλαγή halo points μεταξή δύο διεργασιών που βρίσκονται σε διαφορετικούς επεξεργαστές, διαδικασία που προσεγγίζει σε μεγάλο βαθμό τον αλγοριθμό μας.

Κάνοντας fitting τα δεδομένα που μας επιστρέφει το mpptest (χρόνος επικοινωνίας συναρτήσει του μήκους μυνήματος) με την εξίσωση του ιδανικού μοντέλου επικοινωνίας $T_{msg} = t_s + t_wL$:



Παίρνουμε:
t_s = 82.3 μs
t_w = 0.2 μs

3.1. 1-D διαμερισμός

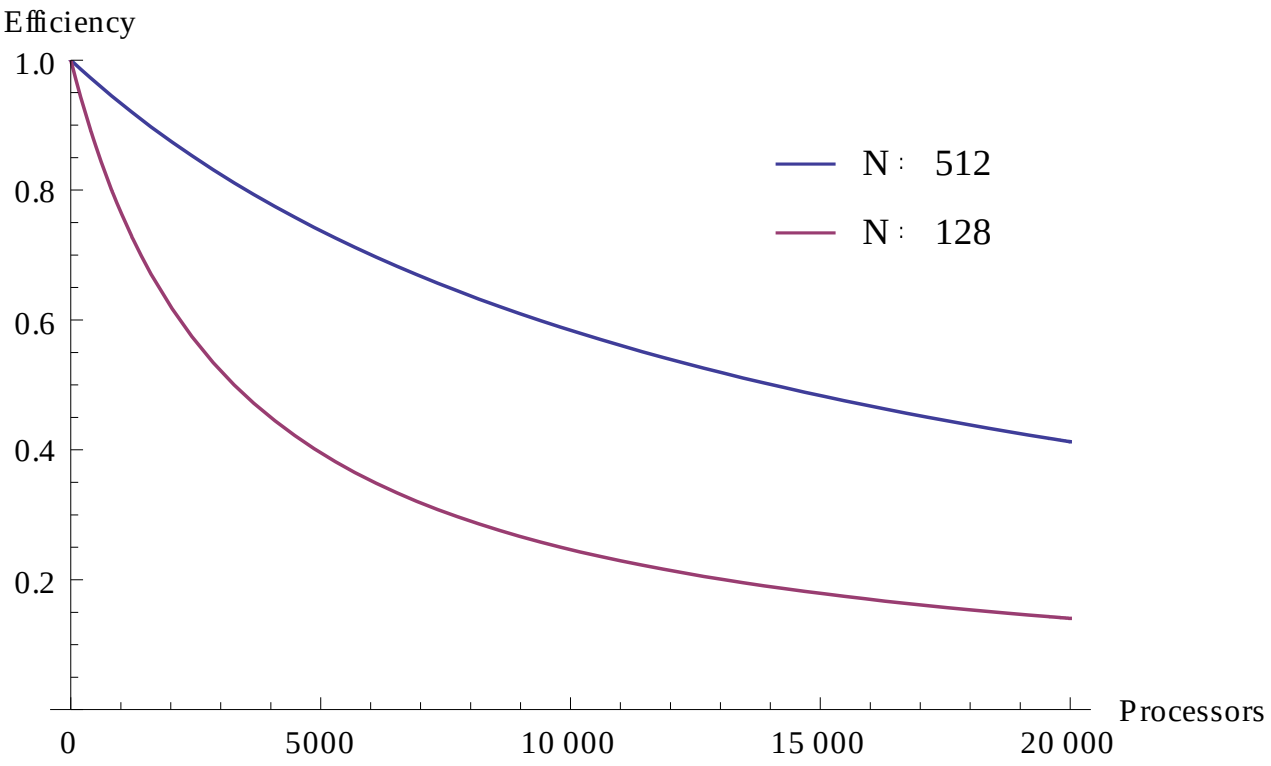
Αν κάνουμε μονοδιάστατο διαμερισμό, τότε κάθε διεργασία θα έχει (N/P) × N × Z σημεία. Κάθε διεργασία θα επικοινωνεί με 2 γείτονες, οπότε ο συνολικός χρόνος που απαιτείται για την επικοινωνία (communication overhead) είναι: $T_{comm} = 2P(t_s + t_w2NZ)$.

$$T_P = t_c \frac{N^2 Z}{P} + t_s 2 + t_w 4NZ$$

$$S = \frac{t_c N^2 Z}{t_c \frac{N^2 Z}{P} + t_s 2 + t_w 4NZ} = \frac{t_c N^2 ZP}{t_c N^2 Z + t_s 2P + t_w 4NZP}$$

$$E = \frac{S}{P} = \frac{t_c N^2 Z}{t_c N^2 Z + t_s 2P + t_w 4NZP}$$

Απο το διάγραμμα P-E παίρνουμε τη κλιμάκωση του 1-D διαμερισμού με τον αριθμό των επεξεργαστών για μεγέθη προβλήματος N = 512 και 128:



3.2. 2-D διαμερισμός

Κάθε διεργασία έχει $\frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}} \times Z$ σημεία του πλέγματος. Κάθε διεργασία επικοινωνεί τώρα με 4 γείτονες.

Το communication overhead θα είναι:

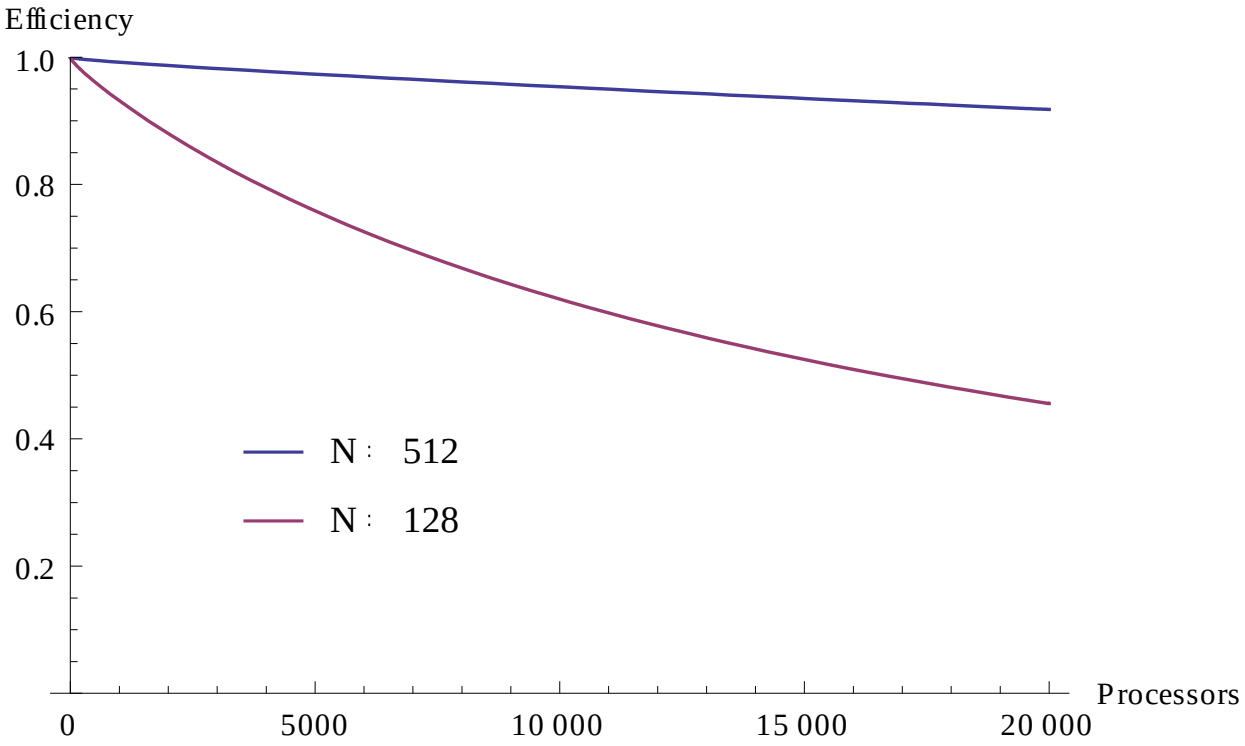
$$t_{comm} = 4P\left(t_s + \frac{t_w 2NZ}{\sqrt{P}}\right)$$

$$T_P = t_c \frac{N^2 Z}{P} + t_s 4 + \frac{t_w 8NZ}{\sqrt{P}}$$

$$S = \frac{t_c N^2 Z}{t_c \frac{N^2 Z}{P} + t_s 4 + \frac{t_w 8NZ}{\sqrt{P}}} = \frac{t_c N^2 ZP}{t_c N^2 Z + t_s 4P + t_w 8NZ\sqrt{P}}$$

$$E = \frac{S}{P} = \frac{t_c N^2 Z}{t_c N^2 Z + t_s 4P + t_w 8NZ\sqrt{P}}$$

Η κλιμάκωση του 2-D διαμερισμού είναι θεωρητικά καλύτερη απο του 1-D:



Έτσι η επιλογή του συγκερασμού σε 2 διαστάσεις που κάναμε επιβεβαιώνεται απο το θεωρητικό μοντέλο ότι είναι η καλύτερη.

4. Υλοποίηση σε MPI – OpenMP

Το επόμενο στάδιο είναι η υλοποίηση του ατμοσφαιρικού μοντέλου σύμφωνα με το σχεδιασμό που έγινε πριν. Τα κύρια σημεία της υλοποίησης είναι τα εξής:

1) Τα σημεία του αρχικού πλέγματος διαμοιράζονται ισόποσα στις διεργασίες, στις 2 διαστάσεις N_x, N_y. Η 3η (κάθετη) διάσταση N_z είναι συγκερασμένη (agglomerated) μέσα σε κάθε διεργασία.

2) Η αντιστοίχηση διεργασιών-επεξεργαστών (mapping) είναι 1-1.

3) Η συνολική μάζα της ατμόσφαιρας διαμοιράζεται ισόποσα στις διεργασίες.

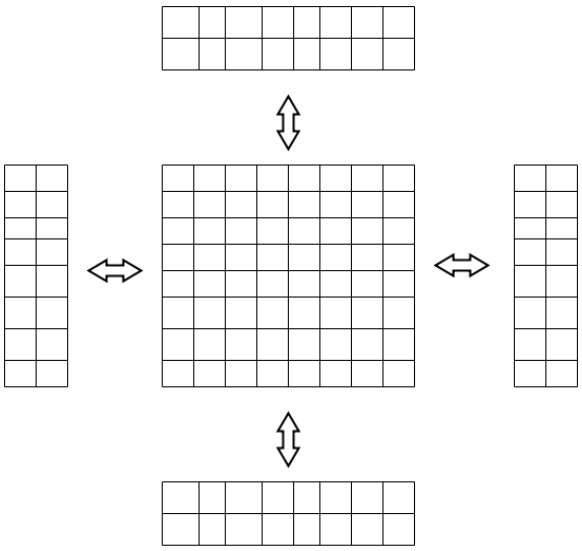
4) Γύρω απο το πλέγμα κάθε διεργασίας αφήνεται χώρος απ' την αρχή για τα συνοριακά σημεία (halo points) που θα ληφθούν απο τους γείτονες. Έτσι αποφεύγονται αργότερα το δυναμικό reallocation και οι συνοριακοί έλεγχοι (boundary checks).

5) Δημιουργούνται datatypes για τα κομμάτια του πλέγματος που πρόκειται να ανταλλάσονται μεταξύ των διεργασιών, ώστε να αποφευχθούν οι άσκοπες αντιγραφές σε buffers.

6) Δημιουργείται ένας cartesian communicator, ο οποίος επιτρέπει την ευκολότερη εύρεση των διεργασιών μέσω των συντεταγμένων τους στο πλέγμα και προσφέρει εν δυνάμει καλύτερη απόδοση αφού το MPI μπορεί να τοποθετήσει γειτονικές διεργασίες σε γειτονικούς επεξεργαστές (rank reordering).

7) Κάθε διεργασία βρίσκει τους 4 γειτονές της στο πλέγμα, εάν υπάρχουν, μέσω των συντεταγμένων τους.

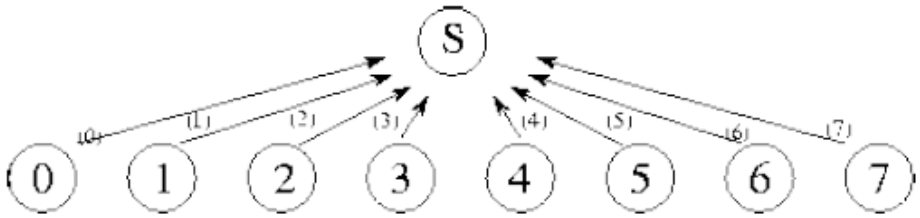
8) Κάθε διεργασία στέλνει τα συνοριακά της σημεία στους γείτονες και λαμβάνει τα δικά τους. Η όλη επικοινωνία είναι ασύγχρονη (non – blocking).



9) Υπολογίζεται το εσωτερικό κομμάτι του πλέγματος της διεργασίας που δεν εξαρτάται απο γείτονες. Επικάλυψη επικοινωνίας και υπολογισμών. Οι υπολογισμοί περιλαμβάνουν το Jacobi finite difference στο οριζόντιο επίπεδο και τη συνάρτηση της ακτινοβολίας στο κάθετο. Τα loops μπορουν προαιρετικά να γίνουν multithreaded με χρήση του OpenMP.

10) Όταν έχουν ληφθεί όλα τα συνοριακά στοιχεία απο τους γείτονες, υπολογίζεται το εξωτερικό εξαρτώμενο κομμάτι. Τα loops μπορουν προαιρετικά να γίνουν multithreaded με χρήση του OpenMP.

11) Κάθε ορισμένες επαναλήψεις εκτελείται ένα global reduction που αθροίζει τις τοπικές μάζες σε όλες τις διεργασίες. Αυτό εδώ γίνεται για να μελετήσουμε πως το global communication επηρεάζει την απόδοση.



12) Τα βήματα 8 – 11 επαναλαμβάνονται για τον καθορισμένο αριθμό επαναλήψεων.

5. Μετρήσεις

Οι μετρήσεις επίδοσης – κλιμάκωσης έγιναν σε 16 μηχανές του εργαστηρίου που ήταν διαθέσιμες. Καθεμία έχει 2 cores, σύνολο 32 cores. Κάθε μέτρηση επαναλαμβάνεται 3 φορές και πέρνεται ο μέσος όρος.

Το πρόγραμμα δέχεται απο command line: το μέγεθος του πλεγματος (N_x, N_y, N_z), τη διάταξη των επεξεργαστών στο επίπεδο, τον αριθμό των επαναλήψεων, και τη συχνότητα που θα γίνεται το reduction.

Ένα παράδειγμα εκτέλεσης για μέγεθος προβλήματος 512 × 512 × 16, σε 32 επεξεργαστές, με 200 επαναλήψεις και reduction να πραγματοποιείται κάθε 10 επαναλήψεις, είναι:

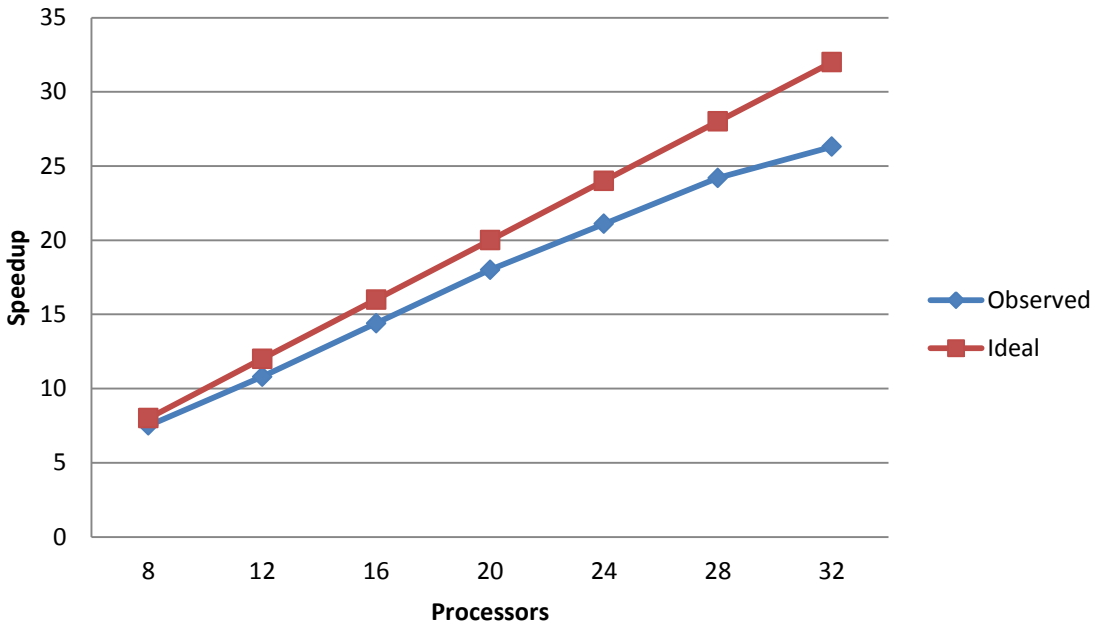
```
>mpiexec -machinefile machines -np 32 AtmModelMPI -nodes 512 512 16 -procs 8 4 –steps 200 -reduce 10
```

Z = 16 στη κάθετη διεύθυνση, σταθερό για όλες τις μετρήσεις. Το reduction είναι προς το παρόν απενεργοποιημένο για να μην επηρρεάζει τις επιδόσεις, η επίπτωσή του θα μετρηθεί παρακάτω. Όλοι οι χρόνοι είναι σε msec.

Ο χρόνος εκτέλεσης T_P, η επιτάχυνση (S) και η αποδοτικότητα (E) του παράλληλου αλγορίθμου, για σταθερό μέγεθος προβλήματος N = 512 και μεταβαλλόμενο αριθμό επεξεργαστών, συγκρίνεται με αυτά που προκύπτουν απο το μοντέλο για τον 2-D διαμερισμό:

P	**Observed T_P**	**Model T_P**	**Observed S**	**Model S**	**E**	**Model E**
8	12584.3	11801.4	7.5	8	0.94	1
12	8710.3	7868.4	10.8	12	0.9	1
16	6529.6	5901.9	14.4	16	0.9	1
20	5250.7	4721.9	18	20	0.9	1
24	4463.8	3935.2	21.1	24	0.88	1
28	3903.1	3373.2	24.2	28	0.86	1
32	3580.7	2951.8	26.3	32	0.82	1

Στο παρακάτω διάγραμμα απεικονίζεται το ιδανικό speedup συναρτήσει του αριθμού των επεξεργαστών σε σχέση με αυτό που παίρνουμε στη πράξη:

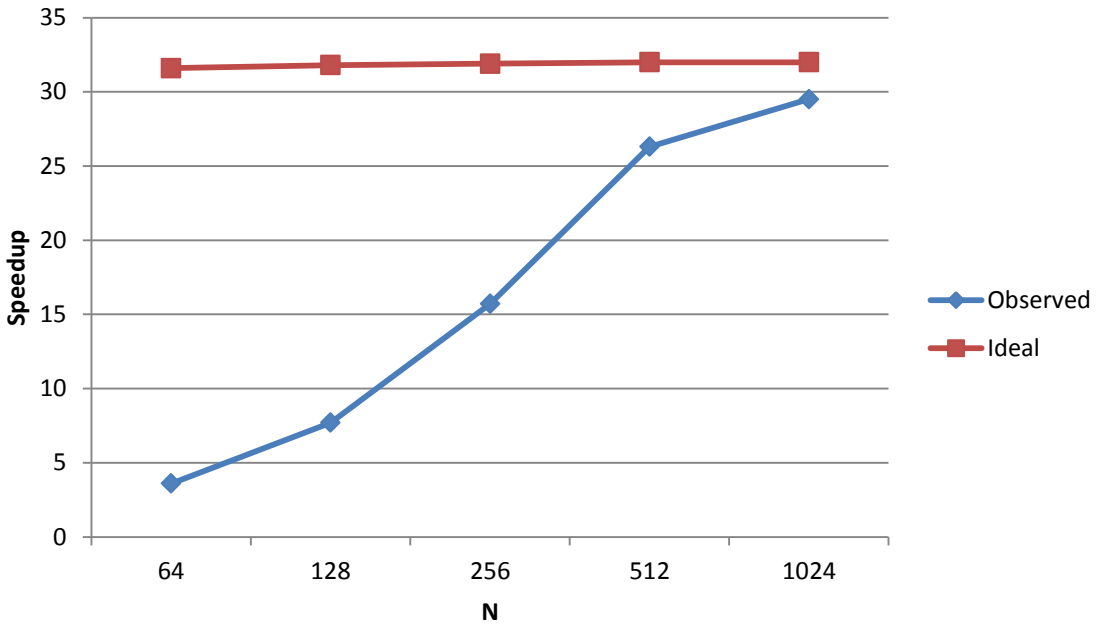


Παρατηρούμε ότι οι χρόνοι εκτέλεσης είναι μεγαλύτεροι απο αυτούς που προβλέπονται απο το μοντέλο. Συνεπακόλουθα, η επιτάχυνση και η αποδοτικότητα είναι μικρότερες απο τις ιδανικές. Αυτό οφείλεται περισσότερο στο ότι το αρχικό μοντέλο δεν είναι ολοκληρωμένο: Κάποια σημεία του αλγορίθμου ή της υλοποίησής του αγνοήθηκαν ως αμελητέα κατα τον ορισμό του μοντέλου, αλλά στη πράξη έχουν εμφανή επίπτωση στο χρόνο εκτέλεσης. Τέτοια μπορεί να είναι: ανισοκατανομή του φόρτου εργασίας (load imbalance) μεταξύ των επεξεργαστών, επανάληψη υπολογισμών (replicated computation), αναντοιστοιχία μεταξύ των εργαλείων και του αλγόριθμου (tool/algorithm mismatch) ο ανταγωνισμός των διεργασιών για το εύρος ζώνης (competition for bandwidth).

Στη συνέχεια θέλουμε να εξετάσουμε τη κλιμάκωση συναρτήσει του μεγέθους του προβλήματος, θέτοντας τον αριθμό των επεξεργαστών ίσο με το μέγιστο (P = 32) και κάνοντας μετρήσεις για διάφορες τιμές του N:

N	Observed T_P	Model T_P	Observed S	Model S	E	Model E
64	414.4	46.7	3.6	31.6	0.11	0.99
128	762.6	185.3	7.7	31.8	0.24	1
256	1496.8	738.8	15.7	31.9	0.49	1
512	3580.7	2951.77	26.3	32	0.82	1
1024	12858.1	11801.4	29.5	32	0.92	1

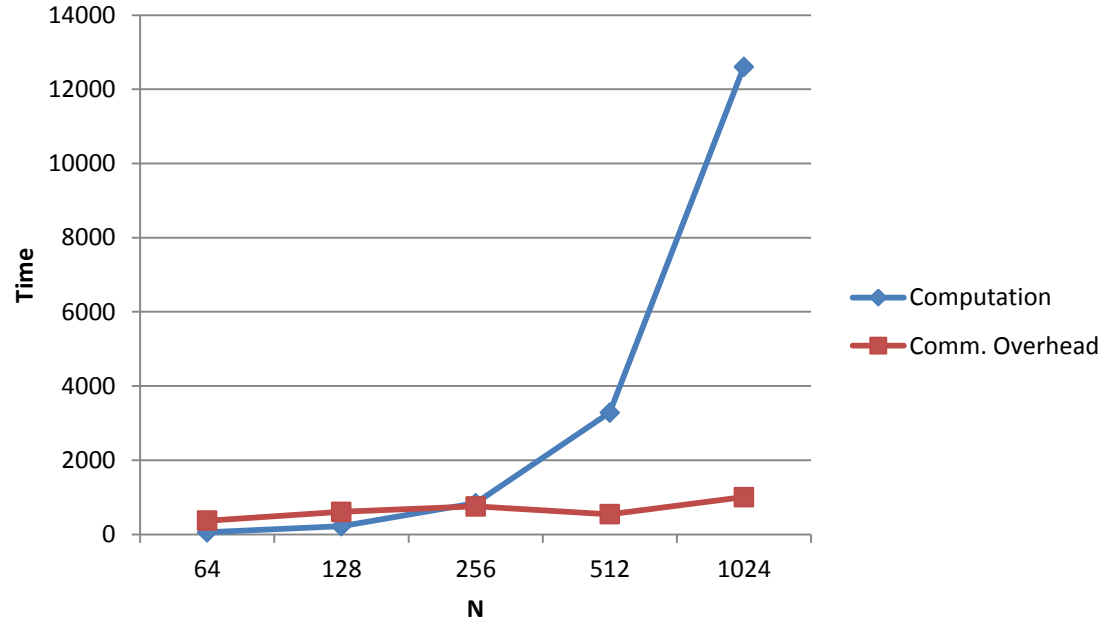
Στο παρακάτω διάγραμμα απεικονίζεται το ιδανικό speedup συναρτήσει του μεγέθους προβλήματος N σε σχέση με αυτό που παίρνουμε στη πράξη:



Παρατηρούμε ότι για μεγέθη προβλήματος N < 256 δεν είναι αποδοτικό να χρησιμοποιήσουμε 32 επεξεργαστές (efficiency = 50% για N = 256), ενώ για μεγέθη N > 512 επιτυγχάνουμε καλή αξιοποίηση των πόρων (efficiency = 82% και 92% για N = 512 και 1024 αντίστοιχα).

Ο λόγος για αυτή τη συμπεριφορά γίνεται φανερός αν δούμε τον καθαρό χρόνο υπολογισμών (computation time) και τον χρόνο που απαιτείται για την επικοινωνία (communication overhead) συναρτήσει του N:

N	Comp. Time	Comm. Time
64	54.9	368.3
128	216.3	607.4
256	838.7	754.8
512	3281.8	543.9
1024	12605.6	1001.6



Για μικρά N, η συμβολή του communication overhead στο συνολικό χρόνο εκτέλεσης είναι σημαντική επηρεάζοντας αρνητικά το efficiency. Για μεγάλα N, η συμβολή του overhead στο χρόνο εκτέλεσης είναι μικρή σε σχέση με τη συμβολή του χρόνου υπολογισμών, οπότε το efficiency είναι καλύτερο. Το overhead θέτει ένα κάτω όριο στο χρόνο εκτέλεσης ενός παράλληλου αλγόριθμου πέρα απο το οποίο δε μπορούμε να πάρουμε καλύτερο efficiency, εκτός αν μειώσουμε το overhead.

Επίπτωση του global reduction στις επιδόσεις

Για 32 επεξεργαστές, μέγεθος προβλήματος N = 512 και 200 επαναλήψεις, μελετούμε την επίπτωση που έχει το reduction στο χρόνο εκτέλεσης ανάλογα τη συχνότητα (κάθε πόσες επαναλήψεις) R_f που πραγματοποιείται:

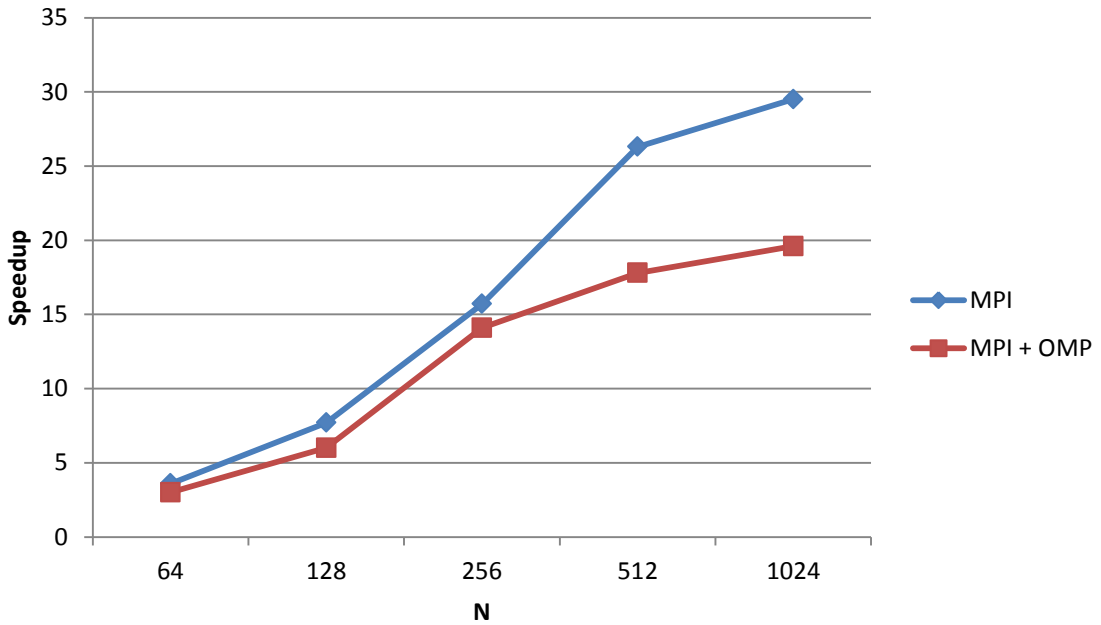
R_f	T_p
0	3536.7
1	3723.4
10	3538.2
50	3515
100	3532

Παρατηρούμε ότι μόνο η πραγματοποίηση του reduction σε κάθε επανάληψη (R_f = 1) έχει μετρήσιμη επίπτωση στις επιδόσεις.

Συνδυασμός MPI – OpenMP

Εκτελούμε το πρόγραμμα με 1 διεργασία ανα μηχανή. Χρησιμοποιούμε το OpenMP για την παραλληλοποίηση των for loops που υπολογίζουν το Jacobi μέσα σε κάθε διεργασία. Συγκρίνουμε το χρόνο εκτέλεσης και την επιτάχυνση με αυτά του σκέτου MPI, για μεταβλητό αριθμό μεγέθους προβλήματος:

	MPI		MPI – OpenMP	
N	T_p	S	T_p	S
64	414.4	3.6	488.8	3
128	762.6	7.7	984	6
256	1496.8	15.7	1672.2	14.1
512	3580.7	26.3	5299.8	17.8
1024	12858.1	29.5	19395.1	19.6

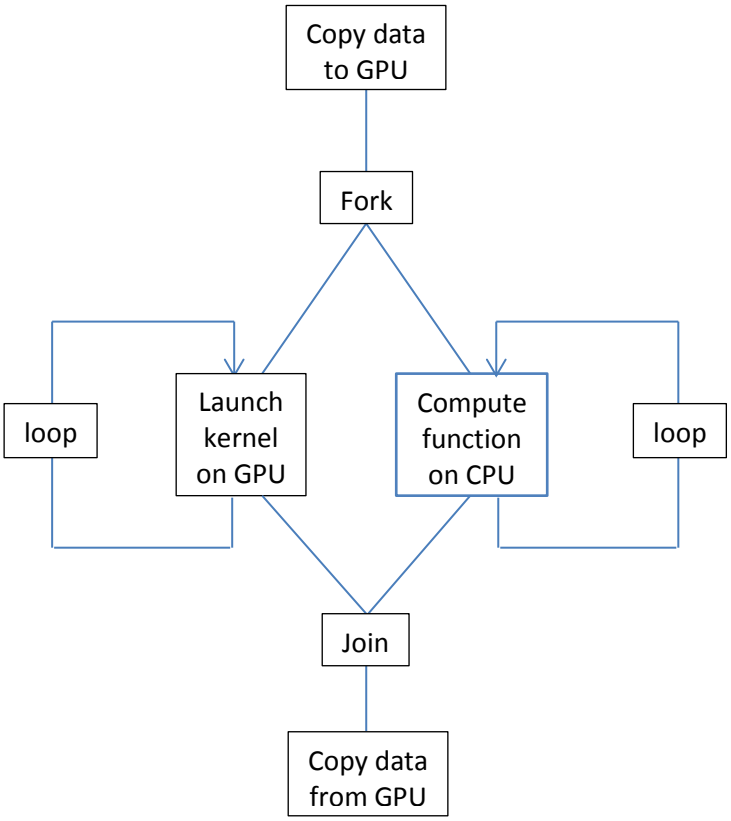


Ο συνδυασμός MPI – OpenMP δεν κλιμακώνει το ίδιο καλά με το σκέτο MPI.

6. Υλοποίηση σε CUDA – OpenMP

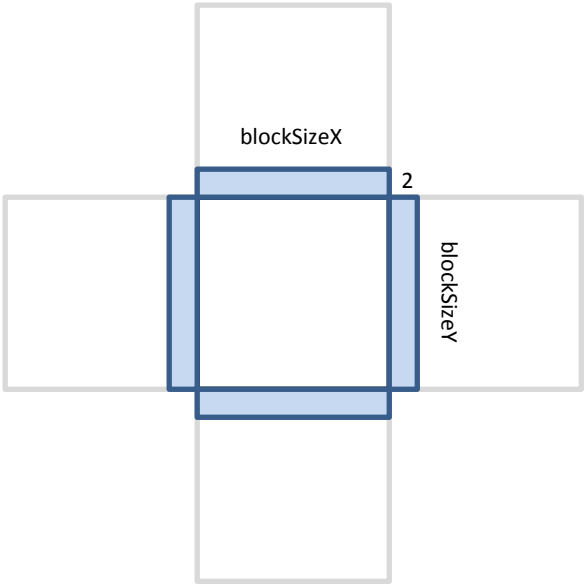
Ώπως είναι γνωστό, ο αλγόριθμος του ατμοσφαιρικού μοντέλου αποτελείται απο 2 μέρη, ένα που υπολογίζει το finite difference που αποτελεί το παράλληλο τμήμα του αλγορίθμου, και ένα που υπολογίζει μια συνάρτηση βάσει της ποσότητας της ηλιακής ακτινοβολίας σε κάθε στρώμα της ατμόσφαιρας, που αποτελεί το σειριακό τμήμα και περιορίζεται στις κάθετες στήλες. Το $1^{ο}$ κομμάτι μπορεί να εκτελεστεί αποδοτικά στη GPU αλλά το $2^{ο}$ όχι. Εφόσον δεν υπάρχουν εξαρτήσεις δεδομένων μεταξύ των 2 κομματιών του αλγόριθμου, μπορούν να εκτελεστούν ανεξάρτητα. Το παράλληλο μέρος (Jacobi finite difference) στη GPU και το σειριακό μέρος (συνάρτηση ακτινοβολίας) στη CPU.

Για τον υπολογισμό του finite difference, η CPU καλεί τον kernel που υπολογιζει ένα time step του finite difference στη GPU τόσες φορές όσος ο αριθμός των επαναλήψεων που έχει οριστεί. Ο υπολογισμός της συνάρτησης γίνεται με τον ακολουθιακο κώδικα στη CPU. Ο πιο απλός τρόπος είναι οι δύο αυτές διαδικασίες να τρέξουν η μία κατόπιν της άλλης. Ένας πιο αποδοτικός τρόπος, που εκμεταλλεύεται και τις δυνατότητες της CPU για παράλληλη εκτέλεση, είναι η δημιουργία 2 νημάτων που το ένα θα είναι υπεύθυνο για την κλήση των kernels και το άλλο για τον υπολογισμό της συνάρτησης. Έτσι οι 2 διαδικασίες μπορούν να ξεκινήσουν και να προχωρούν παράλληλα. Το διάγραμμα της υλοποίησης είναι το παρακάτω:



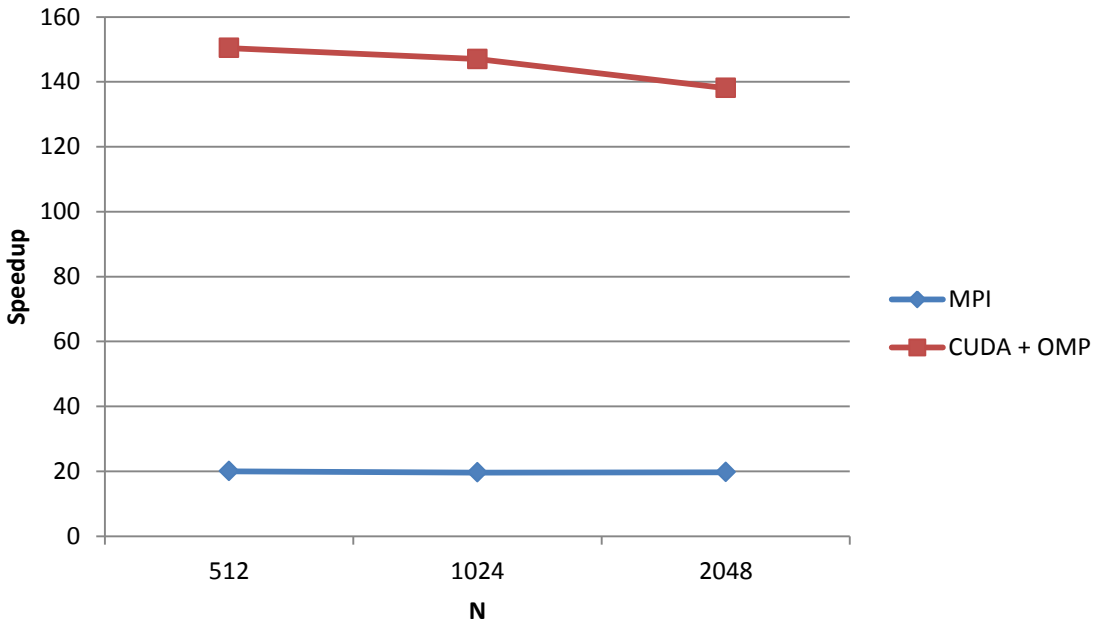
Για τον kernel, το $N_x \times N_y \times N_z$ 3-D πλέγμα του προβλήματος γίνεται mapped σε ένα 3-D grid στη GPU. Κάθε block «φορτώνει» στη shared memory ένα κύβο μεγέθους blockSizeX × blockSizeY × Z απο το πλέγμα που βρίσκεται στη global memory, καθώς και τα στοιχεία απο τους γείτονες (halo

elements) μεγέθους 2 × blockSizeY × Z και blockSizeX × 2 × Z. Κατόπιν υπολογίζεται το finite difference μέσα σε κάθε block, κάθε thread είναι υπεύθυνο για ένα σημείο του πλέγματος.



Η απόδοση για μεγέθη προβλήματος N = 512, 1024, 2048 σε ένα κόμβο με Intel Xeon E8xxx (10 cores/20 threads) και Tesla K40 είναι η παρακάτω. Συγκρίνουμε και με την απόδοση της υλοποίησης MPI που τρέχει σε δύο ίδιους κόμβους (σύνολο 20 cores):

N	Sequential	MPI		CUDA + OpenMP	
	T_s	T_p	S	T_p	S
512	41960.9	2099.4	20	279	150.4
1024	166606.4	8519.9	19.6	1133.6	147
2048	655973.8	33323.1	19.7	4748.7	138.1



Επιτύγχάνουμε κατα μέσο όρο 7.5x καλύτερη επιτάχυνση σε σύγκριση με το MPI.