ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

Ομάδα:

Βασίλειος Μαυρομμάτης - ΑΜ 1115201200106

Ιωσήφ Πονσέτης - ΑΜ 1115201100100

Εισαγωγή

Περιγραφή αντικειμένου της εργασίας:

Σκοπός της εργασίας είναι η βελτίωση ενός προγράμματος MPI, το οποίο προσομοιώνει την μεταφορά θερμότητας πάνω σε μία επιφάνεια.

Η εργασία υλοποιήθηκε σε περιβάλλον Linux-Ubuntu 18.10 LTS. Όλες οι μετρήσεις πραγματοποιήθηκαν στο cluster ARGO, εκτός από τις υλοποιήσεις για CUDA και HIP, οι οποίες δοκιμάστηκαν στις κάρτες γραφικών που αναφέρονται παρακάτω. Για την εργασία έχει πραγματοποιηθεί βελτίωση του αρχικού παράλληλου προγράμματος σε MPI, με εφαρμογή καλύτερης μελέτης, οργάνωσης δεδομένων και σχεδιασμού όσον αφορά στην πλευρά του MPI, όπως και με δημιουργία υβριδικού προγράμματος MPI και OpenMP για μεγαλύτερη απόδοση. Πιο συγκεκριμένα στον παραδοτέο φάκελο της εργασίας υπάρχουν πέντε αρχεία πηγαίου κώδικα σε γλώσσα C, τα οποία είναι:

- ‘mpi_Simple_heat2D.c’ : Περιλαμβάνει κλήσεις συναρτήσεων MPI, καθώς και πολυάριθμες σχεδιαστικές και άλλες επιλογές που επιτυγχάνουν αποδοτικότερη και καλύτερη σχεδιαστικά προσέγγιση του προβλήματος σε σχέση με την αρχική επιλογή.
- ‘mpi_convergence_heat2D.c’ : Περιλαμβάνει την παραπάνω βελτιστοποιημένη προσέγγιση του προβλήματος κάνοντας επιπλέον χρήση της τεχνικής της σύγκλισης (convergence) για ταχύτερη ολοκλήρωση της προσομοίωσης του προβλήματος, με ένα υπολογίσιμο overhead στον χρόνο εκτέλεσης λόγω της Blocking φύσης της σύγκλισης.
- ‘mpi+Openmp_convergence_heat2D’ : Περιλαμβάνει τις παραπάνω βελτιστοποιημένες προσεγγίσεις του προβλήματος κάνοντας επιπλέον χρήση της τεχνολογίας OpenMP, επιτυγχάνοντας μεγαλύτερη παραλληλοποίηση του προβλήματος και συνολικά καλύτερη απόδοση.
- ‘cuda_heat2D.cu’ : Περιλαμβάνει την υλοποίηση του προγράμματος που βρίσκεται στο ‘mpi_Simple_heat2D.c’, γραμμένο σε

Cuda, αξιοποιώντας την κάρτα γραφικών NVIDIA του υπολογιστή. Το παραπάνω πρόγραμμα δοκιμάστηκε στην NVIDIA GeForce GTX 760.

- 'hip_heat2D.cpp' : Περιλαμβάνει την υλοποίηση του προγράμματος που βρίσκεται στο 'mpi_Simple_heat2D.c', γραμμένο σε HIP (ROCM), αξιοποιώντας την κάρτα γραφικών NVIDIA του υπολογιστή, αλλά μπορεί να τρέξει και σε οποιαδήποτε άλλη κάρτα γραφικών. Το παραπάνω πρόγραμμα δοκιμάστηκε στην NVIDIA GIGABYTE GeForce GTX 760 WindForce 3X OC Rev. 2 και στην AMD Asus Radeon RX 580 8GB Rog Strix Gaming OC.

Το compile κάθε αρχείου γίνεται μέσω του script 'compileProgs.sh'.
Τα MPI και OpenMP προγράμματα τρέχουν μέσω των scripts που περιέχονται στο directory 'runScript_ARGO'. Επίσης ο αριθμός των rows και columns είναι defined στον πηγαίο κώδικα των προγραμμάτων, οπότε αν θέλουμε να τρέξουμε για διαφορετικά μεγέθη προβλήματος πρέπει να ξανακάνουμε compile.

Τα CUDA και HIP προγράμματα τρέχουν μέσω των εντολών:
1) ./cudaHeat 160 128 2 4
2) ./hipHeat 160 128 2 4

Όπου για παράδειγμα:

160 είναι ο ριθμός των rows.
128 είναι αριθμός των columns.
2 είναι ο αριθμός των threads που περιέχει κάθε block.
4 είναι ο αριθμός των block που περιέχει το κάθε grid.

Σχεδιασμός Διαμοιρασμού Δεδομένων στις Διεργασίες

Σύμφωνα με την μεθοδολογία Foster κι την φύση του προβλήματος, ο αποδοτικότερος διαμοιρασμός δεδομένων στις διάφορες διεργασίες είναι η μεθοδολογία Block-Block. Η συγκεκριμένη επιλογή προσφέρει καλύτερη κλιμάκωση, καθώς αυξάνεται το μέγεθος του προβλήματος και επιτρέπει αποδοτικότερη διαχείριση των πόρων του συστήματος (πχ. cores).

Επιπλέον η αντιστοίχιση των διεργασιών σε μια καρτεσιανή τοπολογία επιτρέπει στο MPI να αναθέσει τα ranks των διάφορων διεργασιών σε διεργασίες που 'φυσικά' είναι πιο κοντά μέσα στο cluster των διεργασιών (πχ μηχανήματα που είναι στο ίδιο κτήριο και όχι στο διπλανό). Αυτό σε συνδυασμό με χρήση άλλων τεχνικών όπως η Non-Blocking επικοινωνία επιτρέπει μείωση του χρόνου επικοινωνίας και ανταλλαγής μηνυμάτων μεταξύ των διεργασιών με αποτέλεσμα την μείωση του χρόνου της προσομοίωσης του προβλήματος.

Σχεδιασμός MPI κώδικα

Όσον αφορά στην ανάπτυξη του MPI κώδικα ακολουθήθηκαν οι παρακάτω σχεδιαστικές και προγραμματιστικές τεχνικές με σκοπό να μειώσουν όσο το δυνατόν περισσότερο το χρόνο προσομοίωσης του προβλήματος και πιο συγκεκριμένα το χρονικό overhead λόγω διαδιεργασιακής επικοινωνίας, επαναλαμβανόμενων υπολογισμών και εγγραφής και ανάγνωσης μνήμης.

1. Επιλογές επικοινωνίας και επικάλυψη επικοινωνίας με υπολογισμούς:
   Για την πραγματοποίηση της επικοινωνίας μεταξύ των διεργασιών επιλέχθηκε η **Non-Blocking** και **Persistent** communication προσέγγιση.

   Αρχικά με την Non-Blocking προσέγγιση αποφεύγεται ο idle χρόνος που προκύπτει από την εκκίνηση των διαδικασιών αποστολής και λήψης δεδομένων μιας διεργασίας προς μία άλλη γειτονική της, οι οποίες αναστέλλουν την λειτουργία τους ωσότου η γειτονική διεργασία επιβεβαιώσει την εκκίνηση των αντίστοιχων διαδικασιών λήψης και αποστολή. Επίσης, με την χρήση των Non-Blocking συναρτήσεων του MPI όσο οι διεργασίες περιμένουν να λάβουν ή να ολοκληρωθεί η λήψη των δεδομένων τους μπορούν να εκτελέσουν τοπικούς υπολογισμούς, επιτυγχάνοντας έτσι αποδοτικότερη χρήση των πόρων του συστήματος.

   Επιπλέον με τη χρήση Persistent communication επιτυγχάνεται η εξάλειψη του χρόνου επικοινωνίας που οφείλεται σε επανα-υπολογισμό των παραμέτρων των συναρτήσεων αποστολής και λήψης δεδομένων του MPI. Τέτοιες παράμετροι οι οποίες μένουν σταθερές κατά την εκτέλεση του προγράμματος είναι η θέση των γειτόνων (λόγω και της καρτεσιανής τοπολογίας που χρησιμοποιείται μένουν σταθεροί) και τα όρια του προβλήματος της κάθε διεργασίας (υπολογίζονται αρχικά κατά την εκκίνηση του προγράμματος και τον διαμοιρασμό του προβλήματος σε Block).

   Οι συναρτήσεις του MPI που υλοποιούν την παραπάνω λειτουργία είναι οι: MPI_Recv_init(), MPI_Send_init() και MPI_Startall().

2. Datatypes και αποφυγή αναθέσεων:
   Για την αποδοτικότερη αποστολή και λήψη δεδομένων χρησιμοποιήθηκαν τύποι MPI οριζόμενοι από τον χρήστη. Επιπλέον, για μείωση του συνολικού χρόνου εκτέλεσης χρησιμοποιήθηκαν τεχνικές που ελαχιστοποιούν τον

αριθμό των αναθέσεων τιμών και την αντιγραφή δεδομένων σε ενδιάμεσους χώρους αποθήκευσης.

Η δημιουργία custom MPI datatypes, όπως είναι οι τύποι MPI_Type_contiguous() και MPI_Type_vector(), επιτρέπουν την ομαδοποίηση δεδομένων και την αποστολή η λήψη τους με χρήση μίας μόνο συνάρτησης MPI αποστολής η λήψης. Ο χρόνος επικοινωνίας μειώνεται έτσι δραστικά, εφόσον μεγάλος όγκος δεδομένων (στο πρόβλημα μας οι θερμοκρασίες οι οποίες είναι κατανεμημένες σε σειρές και στήλες – rows and columns) μπορεί να σταλεί κάθε φορά σε γειτονικές διεργασίες.

Αποφυγή αναθέσεων επιτυγχάνεται σε όλη την έκταση του προγράμματος με μεταφορά των υπολογισμών, που λαμβάνουν χώρα μία μόνο φορά, έξω από την κύρια επανάληψη που είναι υπεύθυνη για τους υπολογισμούς και τις μετρήσεις του προβλήματος. Ακόμη, χρησιμοποιούνται όπου είναι εφικτό δείκτες, οι οποίοι καθιστούν περιττή την χρήση ενδιάμεσων χώρων αποθήκευσης, εφόσον μπορούμε απλά να αλλάξουμε τη θέση μνήμης που δείχνουν (πχ. πίνακες που κρατούν την τρέχουσα και την προηγούμενη κατάσταση των θερμοκρασιών κάθε διεργασίας). Επίσης γίνεται χρήση αριθμητικής δεικτών, η οποία μας βοηθάει να αναπαραστήσουμε δισδιάστατους πίνακες σε μονοδιάστατους, πράγμα που αυξάνει την ταχύτητα των υπολογισμών και είναι απαραίτητο για την επιτυχή δημιουργία του τύπου οριζόμενου από τον χρήστη: MPI_Type_vector().

3. <u>Τοπολογία διεργασιών</u>
   Για την αποδοτικότερη υλοποίηση του προγράμματος και της επικοινωνίας μεταξύ των διεργασιών χρησιμοποιήθηκε η **Καρτεσιανή Τοπολογία**. Ο λόγος χρήσης Καρτεσιανής τοπολογίας είναι ότι η ίδια η φύση του προβλήματος επιβάλλει πως ο αποδοτικότερος διαμοιρασμός των δεδομένων στις διάφορες διεργασίες είναι η μεθοδολογία **Block-Block**. Έτσι, αν στην παραπάνω προσέγγιση η 'λογική' οργάνωση των διεργασιών του συστήματος αντιστοιχίζεται στην 'φυσική' οργάνωση των διεργασιών (πχ. οι διεργασίες τρέχουν σε μηχανήματα τα οποία βρίσκονται σχετικά κοντά μεταξύ τους – στο ίδιο κτήριο) ο χρόνος που οφείλεται σε διαδιεργασιακή επικοινωνία μειώνεται.

4. <u>Άλλες επιλογές που βελτιώνουν την απόδοση</u>
   Εκτός από τις παραπάνω σχεδιαστικές επιλογές, υλοποιήθηκαν και άλλες non-language και language specific προσεγγίσεις.

   <u>Parallel IO:</u>

Κατά την εκκίνηση του προγράμματος κάθε διεργασία υπολογίζει τα όρια του πίνακα μέσα στα οποία θα μπορεί μόνο αυτή να διαβάζει και να ενημερώνει τιμές και έπειτα δημιουργεί τοπικά το δικό της Block. Όλες δηλαδή οι διεργασίες καλούν την ίδια συνάρτηση αρχικοποίησης, απλά με διαφορετικές παραμέτρους. Ως αποτέλεσμα, στην περίπτωση του **Parallel IO**, δεν υπάρχει ανάγκη για Parallel Read παρά μόνο για Parallel Write. Έτσι, αποφεύγεται η ανάγκη ύπαρξης ενός καθολικού πίνακα, στον οποίο θα έχουν πρόσβαση όλες οι διεργασίες, καθώς και η δημιουργία ξεχωριστού προγράμματος που θα δημιουργεί τον αρχικό πίνακα προς διαμοιρασμό μεταξύ των διεργασιών του συστήματος. Τέλος, αφού κάθε διεργασία ολοκληρώσει τους υπολογισμούς που αφορούν στο δικό της κομμάτι του αρχικού προβλήματος, γράφει με χρήση Parallel Write σε ένα κοινό output αρχείο ('final.dat') τα αποτελέσματα. Αυτό επιτυγχάνεται εσωτερικά με τις συναρτήσεις του Parallel Write οι οποίες επιτρέπουν την ύπαρξη διαφορετικών όψεων (views) του αρχείου για την κάθε διεργασία.

Οι συναρτήσεις του MPI που υλοποιούν την παραπάνω λειτουργία είναι οι: MPI_File_open(), MPI_File_set_view(), MPI_File_write_all(), MPI_File_close().

Παρατήρηση:
Οι παραπάνω συναρτήσεις καλούνται δύο φορές. Μία φορά για το τελικό αρχείο αποτελεσμάτων 'final.dat' και μία για το αρχικό αρχείο 'initial.dat', το οποίο περιέχει όλες τις αρχικοποιημένες θερμοκρασίες της επιφάνειας που υπολογίζονται ξεχωριστά από κάθε διεργασία.

Inline Συναρτήσεις:
Συναρτήσεις που καλούνται πολλές φορές μέσα και έξω από την κύρια επανάληψη του προγράμματος έχουν οριστεί **Inline**, ώστε να αποφευχθεί το overhead που δημιουργείται από επαναλαμβανόμενες κλήσεις της στοίβας (stack) του προγράμματος που τρέχει σε κάθε διεργασία.

Σχεδιασμός CUDA και HIP κώδικα

Όσον αφορά στην ανάπτυξη του CUDA και HIP κώδικα λόγω του γεγονότος ότι η επικοινωνία μεταξύ των threads της κάρτας γραφικών γίνεται μέσω δομών κοινόχρηστης μνήμης (shared memory ή global memory) και όχι μέσω δομών αποστολής και λήψης μηνυμάτων (message passing interface) υπάρχουν διαφορές στον κώδικα υλοποίησης του προβλήματος.

Τα CUDA και HIP threads δεν χρησιμοποιούν την λογική με τα halo points για να υπολογίσουν τα δεδομένα που λαμβάνουν από τους γείτονες τους. Αυτό συμβαίνει διότι όλα τα threads μπορούν ανά πάσα στιγμή να έχουν πρόσβαση σε όλα τα δεδομένα, τα οποία είναι αποθηκευμένα στην global memory. Επίσης, για κάθε βήμα υπολογισμού της θερμότητας, υπάρχουν δύο πίνακες που κρατάνε την παλιά και την καινούργια κατάσταση. Έτσι δεν εμφανίζονται race conditions μεταξύ των threads, αφού όλα τα reads μπορούν να γίνουν ταυτόχρονα στον πίνακα που κρατάει την παλιά κατάσταση, ενώ αντίστοιχα τα writes γίνονται ταυτόχρονα στον πίνακα που κρατάει την καινούργια κατάσταση, με την διαφορά ότι κάθε thread συμπληρώνει τα δεδομένα στο κομμάτι του πίνακα που του αντιστοιχεί.

Η μελέτη κλιμάκωσης πραγματοποιήθηκε στα ίδια μεγέθη προβλήματος με τα προηγούμενα προγράμματα. Χρησιμοποιήθηκαν συνδυασμοί threads/block και blocks/grid μέχρι το σημείο οπού το επέτρεπαν τα χαρακτηριστικά της κάρτας γραφικών (warp_size * multiprocessor_count).

ΑΠΟΤΕΛΕΣΜΑΤΑ – ΜΕΤΡΗΣΕΙΣ

Όλες οι μετρήσεις των προγραμμάτων MPI, MPI+Convergence, MPI+OpenMP και CUDA βρίσκονται στο αρχείο Excel με όνομα ‘Measurements’. Συνολικά έγιναν παραπάνω από τις προτεινόμενες μετρήσεις, πάντα όμως με την προϋπόθεση ότι το μέγεθος του προβλήματος μπορεί να διαμοιραστεί ομοιόμορφα από τους πόρους του συστήματος (nodes, threads, threads/block for CUDA, etc). Κάθε πρόγραμμα έχει πίνακες με μετρήσεις χρόνο, speedup και efficiency. Οι μετρήσεις χρόνου είναι εκφρασμένες σε κλίμακα δευτερολέπτων (resolution: seconds). Επίσης όλα τα προγράμματα επίλυσαν το πρόβλημα για αριθμό 1000 βημάτων (steps).

1. MPI πρόγραμμα χωρίς έλεγχο σύγκλισης.

Οι πίνακες που περιέχουν τις μετρήσεις για το παραπάνω MPI πρόγραμμα είναι αυτοί με το όνομα SIMPLE στο Excel.
Σύμφωνα με τους παραπάνω πίνακες παρατηρούμε ότι:

1. Στην σειριακή εκτέλεση του προγράμματος όσο αυξάνεται το μέγεθος του προβλήματος ο χρόνος εκτέλεσης αυξάνεται σημαντικά.
2. Στις παράλληλες εκτελέσεις του προγράμματος παρατηρούμε ότι για τα μικρά μεγέθη του προβλήματος (80x64, 160x128, 320x256) επιτυγχάνουμε καλύτερο speedup όταν βρισκόμαστε σε 1 υπολογιστικό κόμβο με οποιονδήποτε συνδυασμό από MPI tasks.
3. Αντιθέτως για μεγαλύτερα μεγέθη προβλήματος (640x512, 1280x1024, 2560x2048, 5120x4096) όσο μένουμε στον 1

υπολογιστικό κόμβο το speedup αρχίζει να μειώνεται. Ωστόσο αν αυξήσουμε τον αριθμό των υπολογιστικών κόμβων με οποιονδήποτε συνδυασμό από MPI tasks παρατηρούμε σημαντική βελτίωση του speedup.

4. Από όλες τις περιπτώσεις παρατηρούμε ότι η μεγαλύτερη κλιμάκωση (καλύτερο speedup και efficiency) επιτυγχάνεται για μέγεθος προβλήματος 2560x2048 με χρήση 8 ή 10 υπολογιστικών κόμβων και 64 ή 80 MPI tasks. Επίσης για μεγαλύτερα μεγέθη προβλήματος, όπως το 5120x4096, το speedup σταματάει να βελτιώνεται και το efficiency αρχίζει να πέφτει.

2. MPI πρόγραμμα με έλεγχο σύγκλισης.

Ο έλεγχος σύγκλισης γίνεται κάθε sqrt(steps). Στην περίπτωση μας τα steps είναι 1000, άρα ο έλεγχος σύγκλισης γίνεται κάθε περίπου 31 steps.
Οι πίνακες που περιέχουν τις μετρήσεις για το παραπάνω MPI πρόγραμμα είναι αυτοί με το όνομα SIMPLE CONVERGENCE στο Excel.
Σύμφωνα με τους παραπάνω πίνακες παρατηρούμε ότι:

1. Σε γενικές γραμμές ισχύουν τα ίδια συμπεράσματα και παρατηρήσεις με το MPI πρόγραμμα χωρίς σύγκλιση αφού τα μεγέθη του προβλήματος και οι συνδυασμοί των πόρων του συστήματος είναι ίδιοι.
2. Η μόνη διαφορά που παρατηρείται είναι η κατά λίγο υψηλότεροι χρόνοι εκτέλεσης, οι οποίοι με τη σειρά τους οδηγούν σε χαμηλότερες τιμές speedup και efficiency. Αυτό είναι λογικό λόγω του overhead που προστίθεται λόγω του έλεγχου σύγκλισης των tasks, κάθε 31 steps.

3. MPI + OpenMP πρόγραμμα με έλεγχο σύγκλισης.

Ο έλεγχος σύγκλισης γίνεται κάθε sqrt(steps). Στην περίπτωση μας τα steps είναι 1000, άρα ο έλεγχος σύγκλισης γίνεται κάθε περίπου 31 steps.
Οι πίνακες που περιέχουν τις μετρήσεις για το παραπάνω MPI + OpenMP πρόγραμμα είναι αυτοί με το όνομα HYBRID στο Excel.
Σύμφωνα με τους παραπάνω πίνακες παρατηρούμε ότι:

1. Στο υβριδικό πρόγραμμα MPI + OpenMP παρατηρούμε ότι κάποιοι συνδυασμοί από MPI processes και OpenMP threads έχουν χρόνους εκτέλεσης που ποικίλουν για τον ίδιο αριθμό tasks (processes x threads) και μέγεθος προβλήματος.
2. Πιο συγκεκριμένα παρατηρούμε ότι μικρός αριθμός MPI processes και μεγάλος αριθμός OpenMP threads (δηλαδή συνδυασμοί 1

process – 8 threads) επιτυγχάνουν μικρότερους χρόνους εκτέλεσης για μικρά μεγέθη προβλήματος (80x64, 160x 128, 320x256) και μεγάλους χρόνους εκτέλεσης για μεγάλα μεγέθη προβλήματος (640x512, 1280x1024, 2560x2048, 5120x4096).

3. Αντίθετα παρατηρούμε ότι μεγάλος αριθμός MPI processes και μικρός αριθμός OpenMP threads (δηλαδή συνδυασμοί 4 processes – 2 threads) επιτυγχάνουν μεγαλύτερους χρόνους εκτέλεσης για μικρά μεγέθη προβλήματος (80x64, 160x 128, 320x256) και μικρότερους χρόνους εκτέλεσης για μεγάλα μεγέθη προβλήματος (640x512, 1280x1024, 2560x2048, 5120x4096).
4. Επίσης υπάρχει η μέση περίπτωση (2 processes – 4 threads), η οποία συνολικά έχει και την χειρότερη κλιμάκωση σε σχέση με τις 2 παραπάνω περιπτώσεις, εκτός από συγκεκριμένους συνδυασμούς όπως: 1 node – 2 processes – 4 threads και 10 nodes – 2 processes – 4 threads.
5. Συνολικά παρατηρούμε ότι όσο διαμοιράζουμε τις παραπάνω περιπτώσεις σε περισσότερους υπολογιστικούς κόμβους (nodes) και σε αυξανόμενα μεγέθη προβλήματος, έχουμε βελτίωση του speedup. Αυτό επίσης μπορεί να φανεί και στον αντίστοιχο πίνακα στο Excel όπου τα speedup με τιμή μεγαλύτερη του 1 είναι μαρκαρισμένα με πορτοκαλί. Παρόλα αυτά βλέπουμε ότι καλύτερο efficiency επιτυγχάνουμε όταν χρησιμοποιούμε τον μέγιστο αριθμό υπολογιστικών κόμβων του συστήματος με τον συνδυασμό των 4 MPI processes – 2 OpenMP threads (πάλι μαρκαρισμένα στο Excel με πορτοκαλί).
6. Τέλος, παρατηρούμε ότι το απλό MPI πρόγραμμα με convergence έχει καλύτερες τιμές όσον αφορά στο speedup και κατ' επέκταση στο efficiency σε σχέση με το MPI + OpenMp πρόγραμμα με convergence. Επομένως αν και διαισθητικά περιμέναμε το MPI + OpenMp πρόγραμμα με convergence να έχει καλύτερη απόδοση και χρόνους εκτέλεσης, παρατηρήσαμε ότι λόγω της αρχιτεκτονικής και του διαμοιρασμού των πόρων του συστήματος το αντίστοιχο πρόγραμμα MPI με convergence είχε καλύτερη κλιμάκωση και λιγότερες διακυμάνσεις μεταξύ των συνδυασμών.

4. CUDA και HIP.

Οι πίνακες που περιέχουν τις μετρήσεις για το παραπάνω MPI πρόγραμμα είναι αυτοί με το όνομα CUDA και HIP στο Excel. Να σημειωθεί ότι για τα μεγέθη προβλήματος 2560x2048 και 5120x4096 δεν υπολογίστηκαν χρόνοι εκτέλεσης για λόγους που αναφέρονται παρακάτω.
Σύμφωνα με τους παραπάνω πίνακες παρατηρούμε ότι:

1. Παρατηρούμε ότι οι συνδυασμοί με περισσότερα threads και λιγότερα blocks έχουν καλύτερες αποδόσεις όσον αφορά στον χρόνο εκτέλεσης για το ίδιο μέγεθος προβλήματος.
2. Επίσης παρατηρούμε ότι για σταθερό συνδυασμό από threads και blocks και αυξάνοντας το μέγεθος προβλήματος εμφανίζεται μια γραμμική επιτάχυνση. Στην περίπτωση μας με αυτά τα μεγέθη προβλήματος παρατηρείται γραμμική αύξηση επιτάχυνσης με συντελεστή 4.
3. Το καλύτερο speedup παρατηρείται σε συνδυασμούς των 128 threads και blocks. Αντιθέτως όσο περισσότερα threads και blocks της κάρτας γραφικών χρησιμοποιούμε τόσο περισσότερο μειώνεται το efficiency. Παρόλα αυτά παρατηρούμε ότι για οποιονδήποτε συνδυασμό thread και block (οποιαδήποτε σειρά στον αντίστοιχο πίνακα στο Excel), όσο αυξάνουμε το μέγεθος του προβλήματος το efficiency μεγαλώνει σαν τιμή, πράγμα που δεν συνέβαινε πάντα στις MPI και OpenMP υλοποιήσεις.