Μετρήσεις – Μελέτη Κλιμάκωσης

Γαρμπής Παναγιώτης-Ορέστης 1115201400025
Γκαραγκάνης Ευάγγελος 1115201400033
Κοτρώνης Κωνσταντίνος 1115201400074

-Εισαγωγή:

Η αξιολόγηση του προγράμματος έχει γίνει με βάση τις μετρήσεις του χρόνου εκτέλεσης κάθε υλοποίησης, τόσο για διαφορετικούς επεξεργαστές (1,4,9,16,25), όσο και για διαφορετικά μεγέθη πίνακα (120, 240, 480, … , 15360). Στην υλοποίηση της MPI+OpenMP χρησιμοποιήσαμε ως συνδυασμό διεργασιών και νημάτων τους συνδυασμούς: 1*1, 1*2, 4*2, 9*2, 16*2, που κατά προσέγγιση είναι οι πλησιέστερες δυνατές αντιστοιχίες στις διεργασίες της MPI. Αυτό συμβαίνει επειδή ο τετραγωνικός πίνακας πρέπει να χωριστεί σε ίσα blocks, οπότε μόνο τετράγωνα φυσικών αριθμών μπορούν να χρησιμοποιηθούν για διεργασίες σε πίνακες που οι πλευρές τους είναι πολλαπλάσια των ριζών των διεργασιών. Αντίστοιχα έχουν γίνει και οι μετρήσεις στην υλοποίηση της Cuda, όπου και αντί για πυρήνες, η κλιμάκωση γίνεται με μέγεθος block (1,4,9,16,25) της Gpu για ιδίου μεγέθους πίνακες κελιών.

Λόγω προβλημάτων του extrae στα μηχανήματα της σχολής, δεν έχουμε οπτικοποιήσει τα αποτελέσματα των προγραμμάτων με paraver. Αυτό, από ότι μάθαμε, συνέβη και σε άλλους φοιτητές, σε έναν εκ των οποίων απαντήσατε σε email που έστειλε ότι αν δεν είναι δυνατή η παράθεση των αποτελεσμάτων του paraver, αρκεί η παρουσίαση των μετρήσεων του προγράμματος. Έτσι κι εμείς έχουμε φτιάξει αναλυτικά αποτελέσματα μετρήσεων για πολλές περιπτώσεις σε όλες τις υλοποιήσεις, πλην αυτής με το παράλληλο IO στην MPI, όπου και δεν ζητείται (άλλωστε είναι και κάπως διαφορετική η υλοποίηση των αρχείων εισόδου με την απλή MPI για λόγους ευκολίας, βλέπε README στον φάκελο mpi_parallel_io).

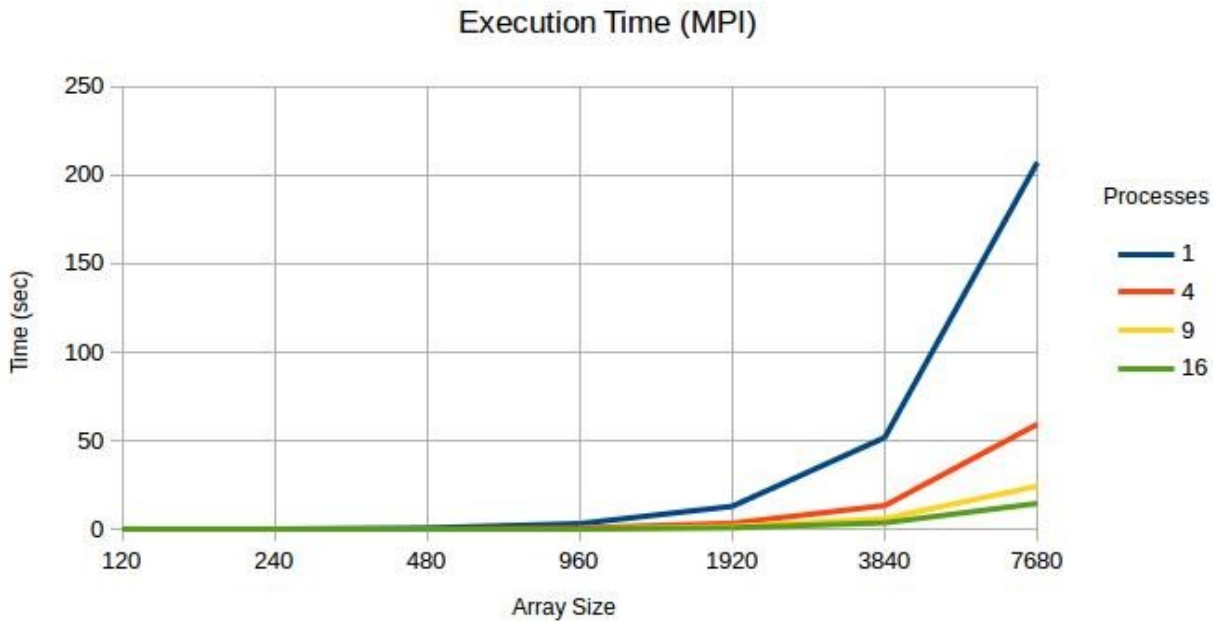
Στον φάκελο που βρίσκεστε (Performance), βρίσκονται 2 αρχεία excel που περιέχουν αναλυτικά τις μετρήσεις των υλοποιήσεων MPI, MPI+OpenMP και Cuda, με διαφορετικές παραμέτρους (εξηγούνται παρακάτω). Επιπλέον, υπάρχει και ένα αρχείο excel όπου δίνονται σε μορφή πινάκων οι χρόνοι εκτέλεσης, οι επιταχύνσεις και οι αποδοτικότητες κάθε συνδυασμού μεγέθους πίνακα με πλήθος πυρήνων. Προφανώς αυτό αναφέρεται στις υλοποιήσεις της MPI και MPI+OpenMP, και όχι σε αυτήν της Cuda. Πάνω σε αυτό το αρχείο βασίζονται και οι παρακάτω απεικονίσεις της κλιμάκωσης των δεδομένων και επεξεργαστών.

Τέλος, όταν ανοίξετε το αρχείο excel που περιέχει τις μετρήσεις των MPI και MPI+OpenMP, θα παρατηρήσετε ότι υπάρχουν μετρήσεις και για 25 διεργασίες (16*2 στην mpi+omp). Την στιγμή που αυτές οι μετρήσεις λαμβάνονταν, δεν υπήρχαν πάνω από 10 υπολογιστές ενεργοί (!), οπότε και προφανώς δεν μπορούμε να δούμε τα αποτελέσματα και την κλιμάκωση γενικότερα αν ο πίνακας διαχωριζόνταν σε 25 blocks (ή 16 blocks με 2 threads στο καθένα). Για τον λόγο αυτό, στις παρακάτω παρουσιάσεις κλιμάκωσης δεδομένων και επεξεργαστών, δεν έχουμε συμπεριλάβει και τις μετρήσεις για 25 διεργασίες (16*2 αντίστοιχα). Παρόλα αυτά έχουμε αφήσει τις μετρήσεις στα αρχεία excel, ώστε να δείξουμε την καθυστέρηση που υπάρχει αν δεν έχουμε την υποδομή σε υλικό για την επιθυμητή παραλληλοποίηση. Επίσης για λόγους ευκρίνειας των γραφημάτων, το ανώτατο μέγεθος πίνακα που χρησιμοποιείται είναι πλευράς 7680 κελιών, ενώ το κατώτατο 120.

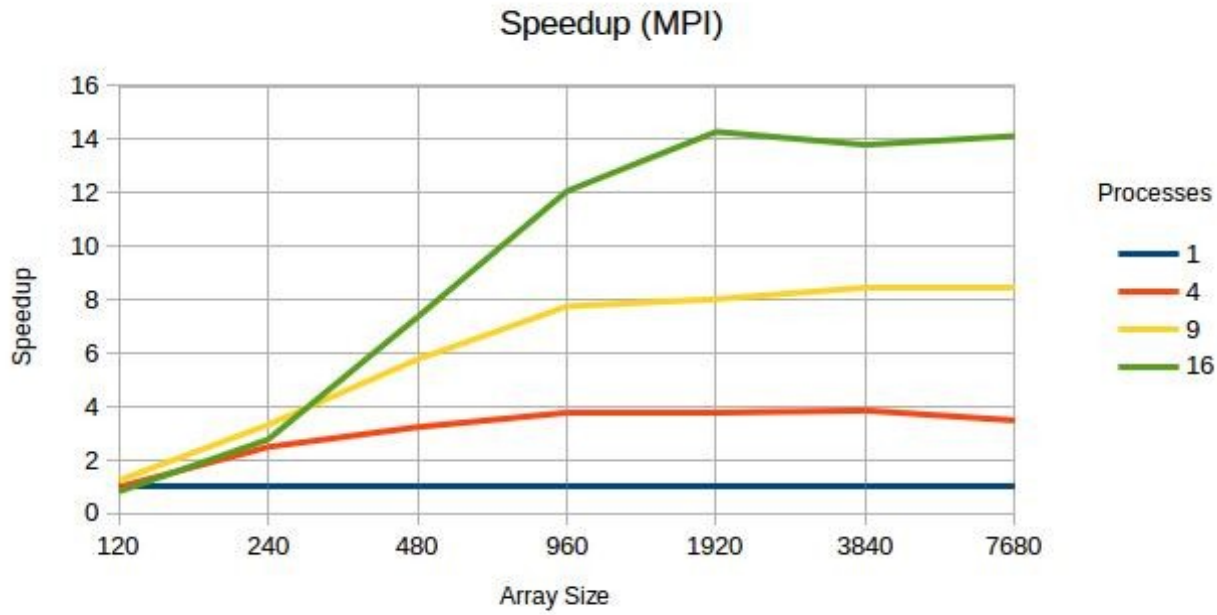
Όλες οι μετρήσεις έχουν γίνει με το αρχείο glider που βρίσκεται στους φακέλους Input Files κάθε υλοποίησης, ενώ οι γεννεές είναι σταθερές (100).

MPI

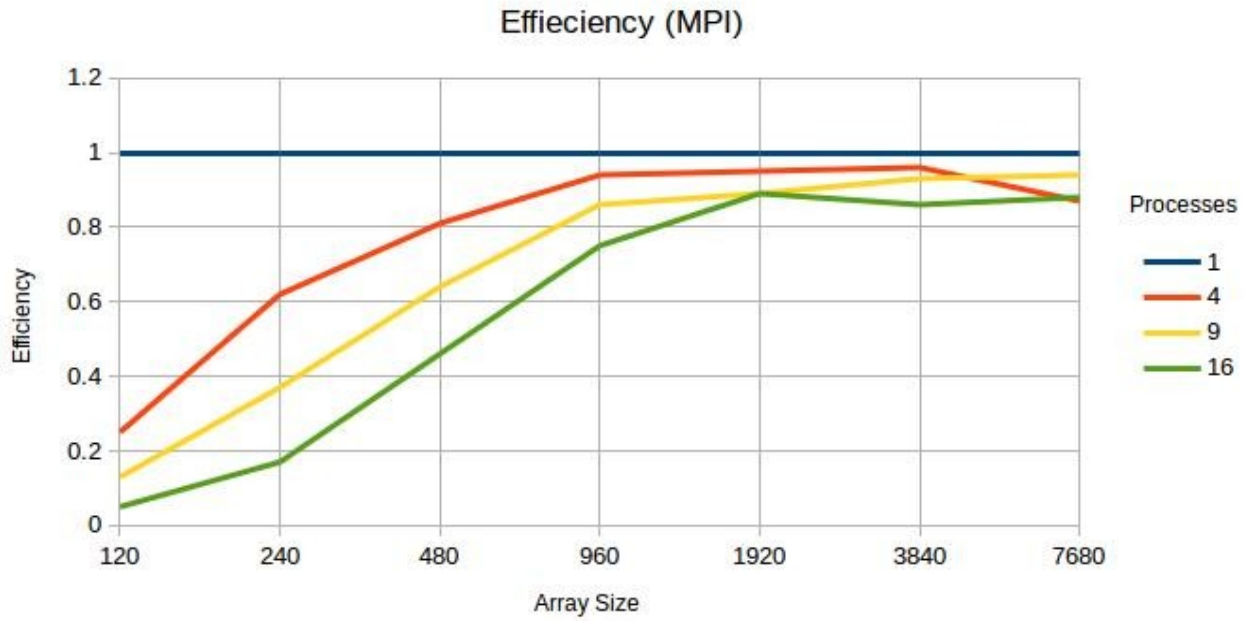
Στο βασικό μας πρόγραμμα της MPI, φαίνονται καθαρά τα αποτελέσματα της κλιμάκωσης των δεδομένων και των επεξεργαστών. Ενώ σε μικρού μεγέθους πίνακες ο χρόνος εκτέλεσης είναι αρκετά μικρός, άρα και οι διαφορές μεταξύ εκτελέσιμων με διαφορετικούς πυρήνες, όσο αυξάνονται οι διεργασίες (δηλαδή τα blocks) είναι αισθητή η διαφορά (ιδίως για τους μεγάλους πίνακες) στον χρόνο εκτέλεσης του προγράμματος. Αν δείτε τους ακριβείς χρόνους στα αρχεία excel, φαίνονται πολύ καθαρότερα οι διαφορές μεταξύ των μικρότερου μεγέθους πινάκων και των αντίστοιχων διεργασιών τους κάθε φορά.



Προφανώς δεν μπορεί να επιταχύνει το πρόγραμμα όσο και οι πυρήνες του (πχ για 16 πυρήνες 16 φορές), καθώς υπάρχουν οι καθυστερήσεις της επικοινωνίας μεταξύ των διεργασιών. Παρόλα αυτά, στα μεγάλα νούμερα, φαίνεται πως πλησιάζει πολύ η πραγματική επιτάχυνση την ιδεατή.



Ενώ σε μικρού μεγέθους πίνακες (120) βλέπουμε ότι οι πολλές διεργασίες έχουν πολύ χαμηλότερη απόδοση από τις λιγότερες (λόγω των περισσότερων επικοινωνιών που βρίσκονται σε εξέλιξη), όσο το μέγεθος του πίνακα αυξάνεται, παρατηρείται το αντίστροφο φαινόμενο (μείωση αποδοτικότητας για τις λίγες διεργασίες, αύξηση για τις πολλές).

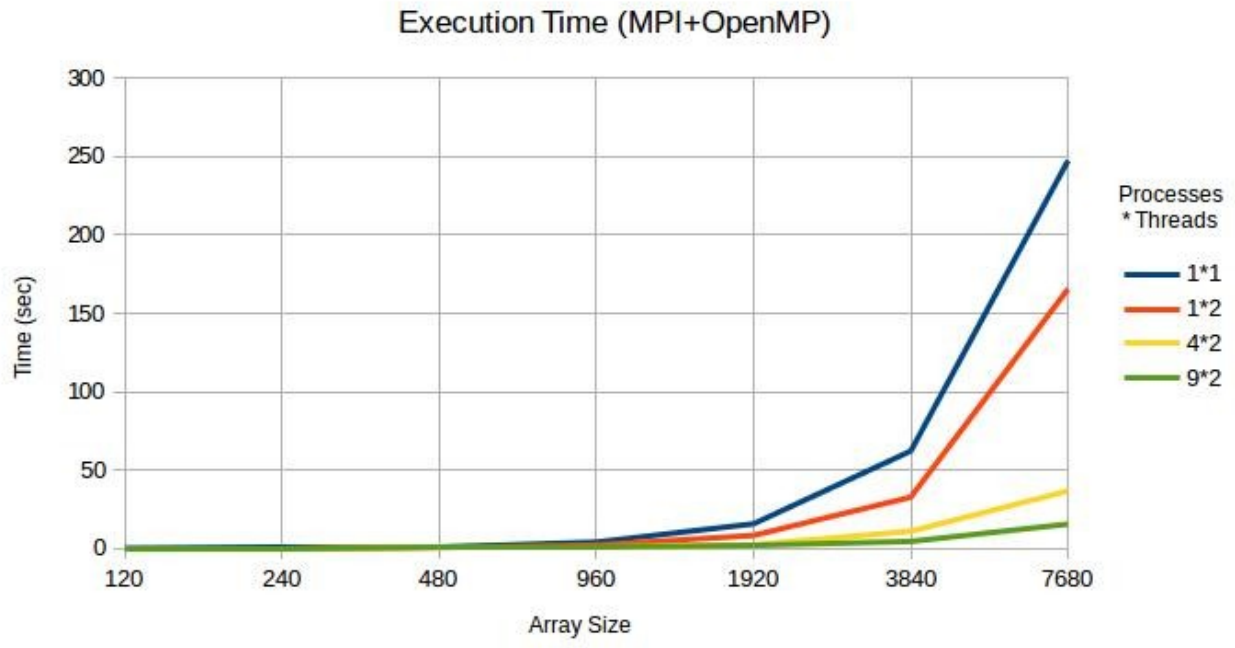


Οι μετρήσεις όπως αναφέρθηκε και προηγουμένως, είναι με 100 επαναλήψεις (γεννεές). Επίσης, μιας και το αρχείο που χρησιμοποιήθηκε ως είσοδο, απεικονίζει το μοτίβο glider, που σε περιοδικό πίνακα συνεχίζει επ’αόριστον την “διαδρομή” του στα κελιά, το πρόγραμμα δεν τερματίζει ποτέ λόγω συνθήκης τερματισμού.

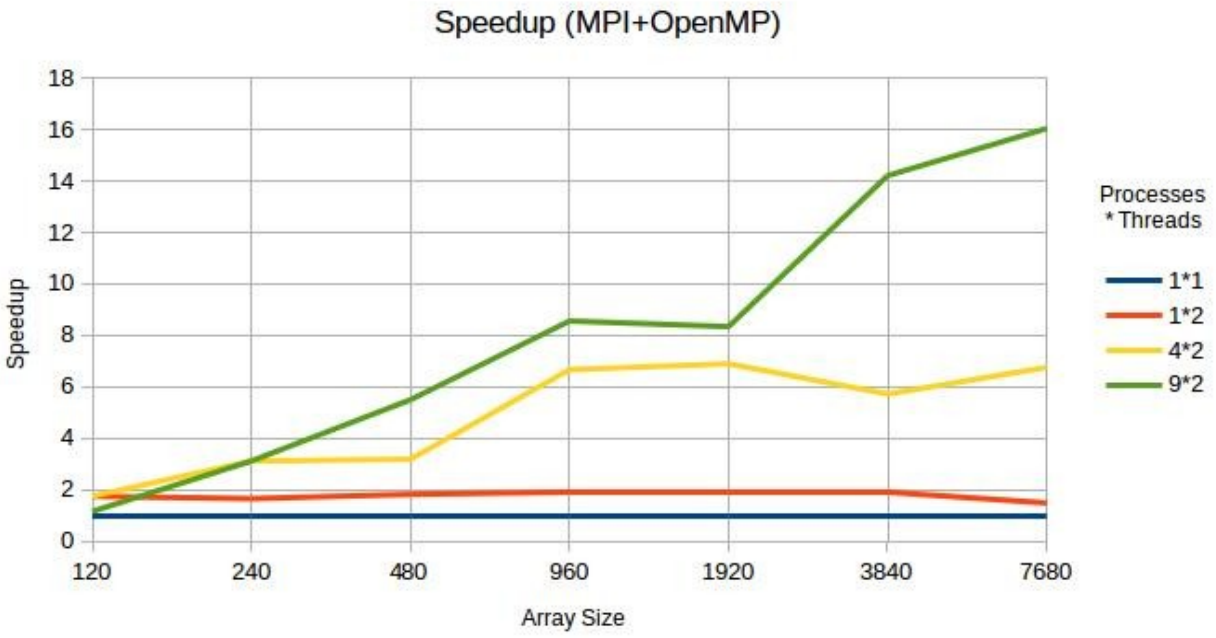
Σε κάθε εκτέλεση που αποτυπώνεται στα παραπάνω γραφήματα υπάρχει έλεγχος τερματισμού. Αν δείτε τις αναλυτικές μετρήσεις των αρχείων excel, τότε, στην συντριπτική τους πλειοψηφία, οι εκτελέσεις με έλεγχο τερματισμού (δηλαδή την χρήση κάποιων MPI_Reduce σε κάθε γεννεά) ήταν λίγο πιο αργές από αυτές χωρίς έλεγχο. Η διαφορά αυτή όμως συνήθως ήταν της τάξης κάποιων δεκάτων του δευτερολέπτου, κάτι λογικό, μιας και οι διεργασίες δεν είναι τόσες πολλές σε αριθμό ώστε να παρουσιαστεί μεγάλη καθυστέρηση στην MPI_Reduce. Για τον λόγο αυτό και οι παραπάνω μετρήσεις εμπεριέχουν έλεγχο τερματισμού, ως πιο αντιπροσωπευτικές.

MPI + OpenMP

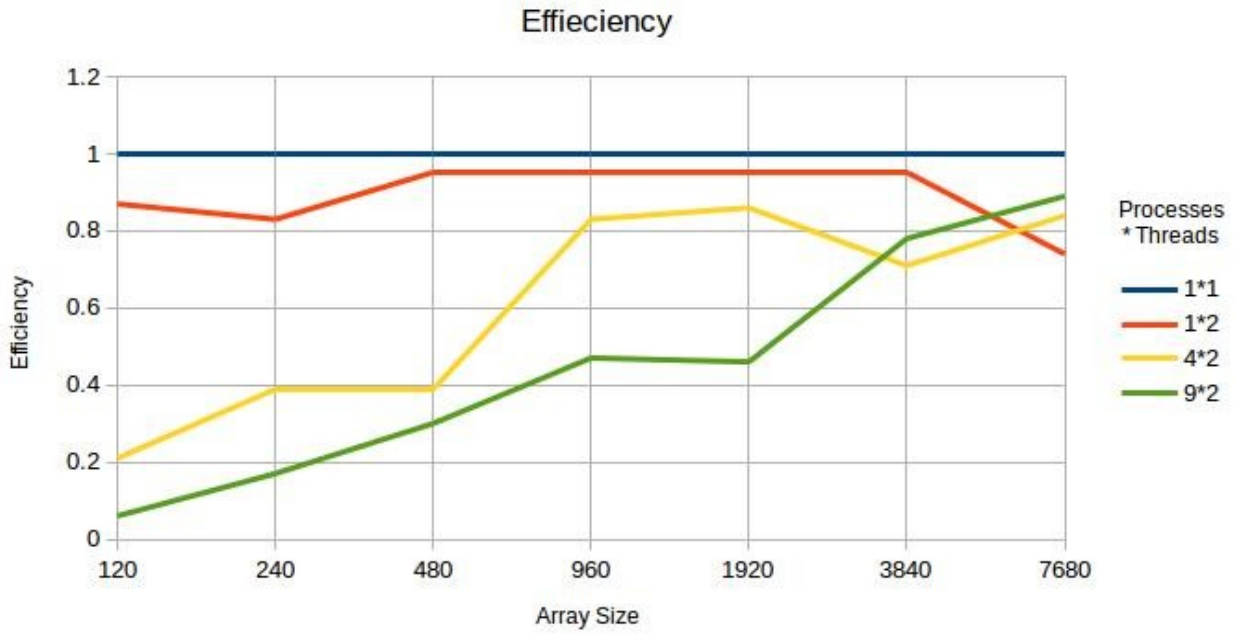
Στην συγκεκριμένη υβριδική υλοποίηση, οι βασικές αρχές που περιγράφησαν για την MPI ισχύουν και εδώ. Στους μικρούς πίνακες οι χρόνοι εκτέλεσης με διαφορετικό πλήθος πυρήνων είναι σχετικά κοντινοί, με μικρές διαφορές, ενώ όσο μεγαλώνει ο πίνακας, αυξάνει και η διαφορά των χρόνων αντιστοίχως. Επίσης και πάλι βλέπουμε αναλογική αύξηση της επιτάχυνσης και της αποδοτικότητας στις πολλές διεργασίες, ενώ μικρή μείωση από ένα σημείο και έπειτα στις λίγες.



Κυρίως λόγω τεράστιων χρόνων εκτέλεσης στους λίγους πυρήνες, αλλά και για ευκρίνεια των πινάκων όπως είπαμε και πιο πάνω, δεν λάβαμε υπόψην μας στην κλιμάκωση πίνακες μεγέθους πλευράς 15360 κελιών. Φαίνεται πως εκεί η υβριδική υλοποίηση είναι αποτελεσματικότερη από την απλή της MPI, καθώς παραλληλοποιούμε το κυριότερο πρόβλημα πολυπλοκότητας που έχουμε, τις εσωτερικές for, κατά την εύρεση γειτόνων στα εσωτερικά κελιά ενός block [$O(n^2)$].



Στις υπόλοιπες υλοποιήσεις, με 7680 κελιά πλευράς και κάτω, υπερισχύει η υλοποίηση με την απλή MPI. Λόγω περιορισμού του υλικού στους υπολογιστές της σχολής, δεν μπορούμε να έχουμε ακριβή αντιστοιχία στις διεργασίες της απλής MPI με τον συνδυασμό διεργασιών και νημάτων σε αυτή την υβριδική μας υλοποίηση. Παρ' όλα αυτά, παρατηρούμε ότι σε γενικές γραμμές, η υλοποίηση με 2 threads και περίπου τους μισούς πυρήνες από ότι η αντίστοιχη της απλής MPI, είναι πιο αργή από την τελευταία. Αυτό παρατηρείται στην σύγκριση αντίστοιχων μετρήσεων χρόνων στο αρχείο excel με τους πίνακες κλιμάκωσης, αλλά και στις αντίστοιχες επιταχύνσεις και αποδοτικότητες (για παράδειγμα είναι εμφανής η διαφορά αποδοτικότητας μεταξύ των 16 πυρήνων της απλής MPI (καλύτερη), και των 9 πυρήνων * 2 νημάτων = 18 παραλληλοποιήσεις της υβριδικής υλοποίησης (αρκετά χαμηλότερη αποδοτικότητα τουλάχιστον από 3840 κελιά πλευράς και κάτω)). Βεβαίως υπάρχει και διαφορά στην απλή σειριακή υλοποίηση, κάτι το οποίο πιθανώς να οφείλεται στις εντολές #pragma του προεπεξεργαστή, αλλά παρά ταύτα δεν παύει να φαίνεται ως καλύτερο στην απόδοση -για αυτές τις τιμές- το απλό πρόγραμμα της MPI.



Όπως και στην απλή υλοποίηση της MPI, έτσι και εδώ χρησιμοποιείται έλεγχος τερματισμού με την βοήθεια της MPI_Reduce και, όπως και πριν, οι διαφορές με τις εκτελέσεις χωρίς έλεγχο είναι μικρές αλλά εμφανείς για την επίδειξη της -έστω και μικρής- καθυστέρησης.

Επιπλέον, η κλιμάκωση εδώ δεν είναι τόσο καθαρή όσο στην MPI, καθώς 1-2 χρόνοι δεν ήταν και απολύτως συμβατοί με την συνάρτηση κλιμάκωσης. Αυτό μάλλον οφείλεται σε πιθανή χρήση κάποιου υπολογιστή και από άλλο άτομο κατά την διάρκεια των μετρήσεων, κάτι που καθυστερεί την υλοποίηση του προγράμματος μας, αφού θα έτρεχαν και άλλα εκτελέσιμα ταυτοχρόνως με ίδια, ή και μεγαλύτερη ακόμη προτεραιότητα. (π.χ firefox).

Cuda

Στην υλοποίηση της Cuda, η κλιμάκωση γίνεται με διαφορετικό μέγεθος blocksize. Επιπλέον του ελέγχου τερματισμού, εδώ παραθέτουμε και την σύγκριση της υλοποίησης Cuda με ή χωρίς shared memory. Σε γενικές γραμμές, όπως φαίνεται και στις μετρήσεις των χρόνων, χρησιμοποιώντας shared memory έχουμε καλύτερα αποτελέσματα, καθώς εκμεταλλευόμαστε την ταχύτατη μνήμη της κάρτας γραφικών. Επίσης σε 2 υλοποιήσεις, μετράμε και την διαφορά του χρόνου κατά τον έλεγχο περιοδικότητας σε επεξεργαστή της gpu ή στον κύριο και ταχύτερο της cpu. Όπως αναμενόταν, στους μικρούς πίνακες συμφέρει η χρήση της ταχύτερης cpu, ενώ στους μεγάλους συμφέρει σε κάποιο thread της gpu, καθώς αποφεύγουμε τις μεγάλες αντιγραφές από και προς την cpu. Επιπλέον εδώ γίνεται ο έλεγχος τερματισμού με το μοτίβο boat, που όντως πρέπει να τερματίζει. Έτσι, μετά από 10 γεννεές, οπότε και γίνεται ο πρώτος έλεγχος, το πρόγραμμα τερματίζει. Μιας και ο αριθμός επαναλήψεων είναι σταθερός (100), οι χρόνοι πρέπει να είναι περίπου οι υποδεκαπλάσιοι από αυτούς χωρίς έλεγχο τερματισμού, κάτι που όπως βλέπουμε και στις μετρήσεις στο αντίστοιχο αρχείο excel ισχύει.

Size NxN / Parameters	/wo Shared Memory [-s 0]	/w Shared Memory [-s 1]
	Without Terminal Checking [-d 0] | | With Terminal Checking [-d 1]	*Without Terminal Checking [-d 0] | | With Terminal Checking [-d 1]*
120 x 120	**[-p 0] (b 16) :** 0.004435 **[-p 1] (b 16) : 0.008323**	**[-p 0] (b 16) :** 0.004272 **[-p 1] (b 16) :** 0.007727
Blocksize [1 / 4 / 9 / 16 / 25]	[0.016008 /0.008529/ 0.008091/ 0.008368/0.008967] || [0.002353 / 0.001331 /0.001376/ 0.001508 /0.001589]	[NaN (needs >=4 blocks)/0.010748/ 0.007886/0.007666/0.007985] || [NaN (needs >=4 blocks)/ 0.001804/0.001298/0.001457/0.001494]
240 x 240		
Blocksize [1 / 4 / 9 / 16 / 25]	[0.053392/0.017183/ 0.015840/0.016743/0.018793] || [0.006422 / 0.002603 / 0.002427/0.002378/0.002612]	[NaN (needs >=4 blocks)/0.028332/0.015536/0.015510/0.015924] || [NaN (needs >=4 blocks)/ 0.003670/ 0.002385/0.002193/ 0.002447]
480 x 480		
Blocksize [1 / 4 / 9 / 16 / 25]	[0.194260/ 0.041643/ 0.035478/0.039434 / 0.046221] || [0.022352/0.005604/ 0.004906/0.005085/0.005874]	[NaN (needs >=4 blocks) / 0.087998/ 0.034948/0.033581/ 0.035863] || [NaN (needs >=4 blocks)/ 0.010318/ 0.004491/0.004480/ 0.004745]
960 x 960		
Blocksize [1 / 4 / 9 / 16 / 25]	[0.744836/ 0.129786/0.103559/0.119270/0.143268] || [0.084035/ 0.015827/ 0.013020/0.014626/0.017270]	[NaN (needs >=4 blocks)/0.316586/ 0.097745/0.095080/0.105169] || [NaN (needs >=4 blocks)/ 0.036597/ 0.012265/0.011521/0.013094]
1920 x 1920		
Blocksize [1 / 4 / 9 / 16 / 25]	[2.864783/0.390376/0.281987/0.344215/0.442515] || [0.321129/ 0.046197/0.034231/0.041031/0.052039]	[NaN (needs >=4 blocks)/ 1.138525/ 0.261773/ 0.249308 /0.289718] || [NaN (needs >=4 blocks)/0.128774/0.031808/0.030021/0.035009]
3840 x 3840		
Blocksize [1 / 4 / 9 / 16 / 25]	[11.295897/ 1.315427/0.889017/1.128568/1.539613] || [1.263866 / 0.154845/ 0.107626/0.133188/0.179107]	[NaN (needs >=4 blocks)/4.319182/ 0.793930/0.741791/ 0.910018] || [NaN (needs >=4 blocks)/0.488410/ 0.094559/0.090423/0.108643]
7680 x 7680		
Blocksize [1 / 4 / 9 / 16 / 25]	[47.179214/5.072776/3.102343/4.041420 /5.699138] || [5.254525/0.585446/0.370238/0.469981/ 0.657926]	[NaN (needs >=4 blocks)/17.438341/2.739638/2.465086/3.143569] || [NaN (needs >=4 blocks)/1.958414/0.324843/0.296304/0.371881]
15360 x 15360	**[-p 0] (b 16) :** 22.279245 **[-p 1] (b 16) :** 15.338717	**[-p 0] (b 16) :** 15.853344 **[-p 1] (b 16) :** 8.937415
Blocksize [1 / 4 / 9 / 16 / 25]	[207.492691/ 20.474800/12.355147/15.344153 /21.525053] || [22.012371 /2.359265/1.398143/ 1.782702/2.464107]	[NaN (needs >=4 blocks)/71.392441/10.273337/8.931963/1.711120] || [NaN (needs >=4 blocks)/1.961762/1.218280/ 1.068194/1.010859]
30720 x 30720		
Blocksize [1 / 4 / 9 / 16 / 25]	[Nan/NaN/Nan/Nan/Nan] || [NaN/NaN/Nan/Nan/Nan]	[NaN (needs >=4 blocks)/Nan/NaN/Nan / Nan] || [NaN (needs >=4 blocks)/Nan/NaN/Nan / Nan]

Σε σχέση με τις μετρήσεις στην MPI, βλέπουμε ότι χρησιμοποιώντας την cuda πετυχαίνουμε πολύ καλύτερους χρόνους, κάτι που άλλωστε είναι λογικό, καθώς η παραλληλοποίηση πηγαίνει σε κάθε κελί και όχι μόνο στα blocks. Βεβαίως, τα blocksize δεν έχουν την ίδια εναλλαγή με αυτή των διεργασιών στην MPI, καθώς εδώ απλά χωρίζει με διαφορετικούς τρόπους τα blocks της gpu, επομένως και τα γειτονικά threads. Παρατηρούμε ότι σε γενικές γραμμές, η καλύτεροι χρόνοι επιτυγχάνονται με μέγεθος block 9.