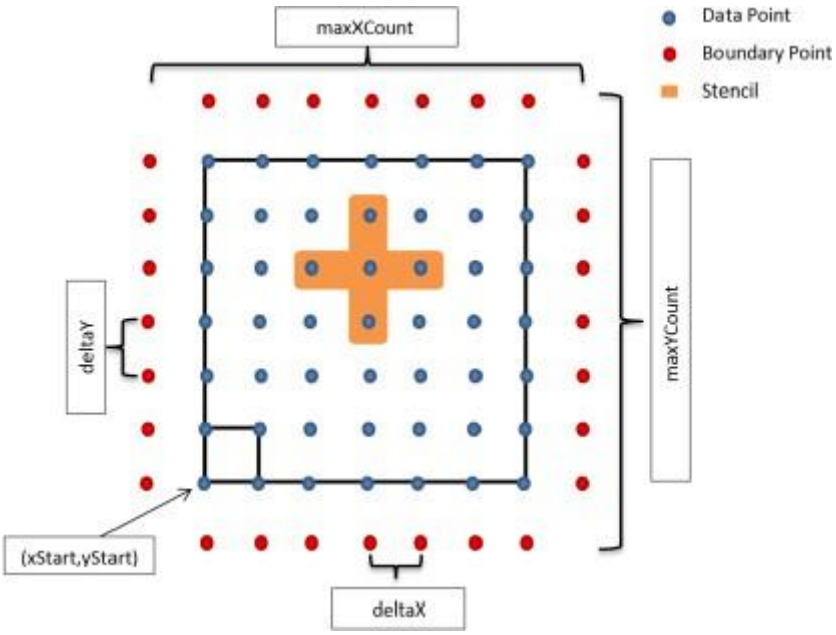
Jacobi Method with successive over-relaxation

Εργασία χειμερινού εξαμήνου ακαδημαϊκού έτους 2020-2021

Παράλληλα Συστήματα



Πέτρος Μπακόλας

Γιάννης Φώτης

Περιεχόμενα:

Εισαγωγή

Η εργασία αυτή αφορά την ανάπτυξη, τη μελέτη και την αξιολόγηση σειριακού και παράλληλου κώδικα για τον υπολογισμό της εξίσωσης Poisson μέσω της μεθόδου Jacobi με διαδοχική ανοχή σφάλματος.

Έχουμε συμπεριλάβει 3 διαφορετικές υλοποιήσεις.

- Ακολουθιακή βελτιωμένη λύση
- Παραλληλοποιημένη λύση με MPI (1 αρχείο με Allreduce και 1 χωρίς)
- Υβριδική παραλληλοποιημένη λύση με MPI & OpenMp

Στο παραδοτέο υπάρχουν συνολικά 4 φάκελοι. Οι τρεις αφορούν τις 3 παραπάνω υλοποιήσεις, ενώ ο τέταρτος (Measurements) περιέχει όλες τις μετρήσεις (αρχεία .o και .mpiP) που κάναμε καθώς και το αντίστοιχο αρχείο excel με τους 3 πίνακες χρόνων, επιτάχυνσης και απόδοσης, αλλά και το γράφημα αυτών των δεδομένων για κάθε μία απ' τις υλοποιήσεις μας.

Εκτέλεση: Σε κάθε φάκελο έχουμε το αντίστοιχο makefile και όλα τα απαραίτητα PBSscript (για 1, 4, 9, 16, 25, 36, 49, 64, 80 διεργασίες ή 1, 4, 16, 36, 54, 80 νήματα)

Ακολουθιακό Πρόγραμμα

Οι αλλαγές που πραγματοποιήσαμε στο δοσμένο πρόγραμμα ήταν οι εξής:
Οι συναρτήσεις «one_jacobi_iteration» και «checkSolution» έχουν καταργηθεί και το σώμα τους υλοποιείται μέσα στο σώμα της main συνάρτησης, στα σημεία όπου γινόταν πριν η κλήση αυτών των δύο. Επιπλέον τροποποιήσαμε σημεία του κώδικα όπου γίνονταν πολλές αναθέσεις, αντικαθιστώντας αυτές τις εντολές με μία μεγάλη παράσταση η οποία χρειάζεται μόνο μία ανάθεση.
Η μεγαλύτερη ωστόσο βελτίωση στην απόδοση του προγράμματος προήλθε απ' την αποθήκευση των τιμών fX και fY σε δύο πίνακες double. Αυτό το κάναμε, γιατί παρατηρήσαμε ότι και στις 50 επαναλήψεις, οι τιμές αυτές παραμένουν σταθερές και επομένως αρκεί να τις υπολογίσουμε μόνο την πρώτη φορά.
Με αυτές τις αλλαγές, μειώσαμε συνολικά τον ακολουθιακό χρόνο κατά 39% ή πιο συγκεκριμένα, για πρόβλημα διάστασης 840*840, καταφέραμε μείωση από 0,837s στα 0,509s.

Σχεδιασμός MPI Παραλληλοποίησης

1. Δημιουργήσαμε την τοπολογία μας ως καρτεσιανό σύστημα διεργασιών. Για διεργασίες 1, 4, 9, 16, 25, 36, 49, 64 χωρίζουμε τον αρχικό πίνακα στον αντίστοιχο αριθμό μπλοκ με (√n x √m) στοιχεία έκαστο. Για 80 διεργασίες, σπάμε τα δεδομένα σε 80 ίσα μπλοκ, με το κάθε ένα απ' αυτά να έχει (n/8 x m/10) στοιχεία. Σε κάθε περίπτωση, όπου n και m, θεωρούμε τις διαστάσεις του αρχικού προβλήματος.

2. Δημιουργήσαμε το καρτεσιανό grid διεργασιών και τις κάναμε reorder με τη χρήση της συνάρτησης «MPI_Cart_create». Έπειτα, γνωρίζοντας τις συντεταγμένες κάθε διεργασίας στο πλέγμα, υπολογίσαμε τα xBlockDimension και yBlockDimension, που αναφέρονται στο μήκος και το ύψος του υποπίνακα που αναλαμβάνει μία διεργασία αντίστοιχα. Και εδώ, αν οι διεργασίες είναι τετράγωνο κάποιου αριθμού, τότε xBlockDimension = √comm_sz και yBlockDimension = √comm_sz, ενώ αν οι διεργασίες είναι 80, τότε xBlockDimension = n/8 και yBlockDimension = m/10. Και πάλι, ως n, m εννοούμε τις διαστάσεις του αρχικού πίνακα και comm_sz το πλήθος των διεργασιών.
3. Με βάση τον υπολογισμό αυτών των δύο μεταβλητών, η κάθε διεργασία μπορεί να υπολογίσει πόσο χώρο χρειάζεται να δεσμεύσει για τον τοπικό της πίνακα δεδομένων αλλά και να υπολογίσει τα local_x και local_y της, που χρειάζεται για το σωστό υπολογισμό των fX και fY της στη συνέχεια.
4. Καλείται η «MPI_Cart_shift» για να αναγνωρίσει κάθε διεργασία τις γειτονικές της διεργασίες.
5. Ορίζονται 2 δομές δεδομένων MPI, συγκεκριμένα η row (τύπου contiguous) και η column (τύπου vector), οι οποίες θα χρησιμοποιηθούν στη συνέχεια για την αποστολή και τη λήψη ολόκληρων σειρών και στηλών δεδομένων αντίστοιχα.
6. Η επικοινωνία που υλοποιήσαμε είναι απλή, και όσο πιο μικρή γίνεται, με χρήση halo swap. Ο πίνακας της κάθε διεργασίας επεκτείνεται κατά 2 σειρές και δύο στήλες, ώστε να αποθηκευτούν τα halo points. Κάθε διεργασία στέλνει τα στοιχεία που βρίσκονται στην δεύτερη και προτελευταία σειρά και στήλη, ενώ τα στοιχεία που λαμβάνει αποθηκεύονται στην πρώτη και τελευταία σειρά και στήλη (halo).
7. Ανάλογα την περίπτωση του κώδικα (υπάρχουν 2 αρχεία mpi), στη μία περίπτωση τα δεδομένα κάθε διεργασία συγκεντρώνονται μέσα στην κεντρική επανάληψη με τη χρήση της «MPI_Allreduce» από όλες τις διεργασίες ή με τη χρήση της «MPI_Reduce» έξω από την επανάληψη και μόνο απ’ τη διεργασία 0, στο τέλος.
8. Έχουμε παραλληλοποιήσει και το τμήμα κώδικα της «CheckSolution», μιας και τώρα κάθε διεργασία υπολογίζει το δικό της σφάλμα με την επαναληπτική μέθοδο και στη συνέχεια αυτά συγκεντρώνονται στη διεργασία 0 αποκλειστικά με τη χρήση της «MPI_Reduce». Ωστόσο, παρόλο που ο ακολουθιακός κώδικας δεν υπολογίζει το χρόνο του υπολογισμού του σφάλματος της επαναληπτικής μεθόδου στον προσμετρούμενο χρόνο, επιλέξαμε συνειδητά να το λάβουμε υπόψιν στο παράλληλο πρόγραμμα. Δηλαδή οι μετρήσεις αφορούν και το χρόνο που αφιερώνει το πρόγραμμα για τον υπολογισμό μέσω της μεθόδου Jacobi αλλά και για την επαναληπτική μέθοδο, μιας και αυτή υπολογίζεται πλέον από πολλές διεργασίες/νήματα. Αν επιλέγαμε να μετράμε μόνο την κύρια επανάληψη, ο χρόνος του προγράμματος θα ήταν ελάχιστα μικρότερος, όπως φάνηκε από δοκιμές που κάναμε με αυτό τον τρόπο. (στα αρχεία .o)

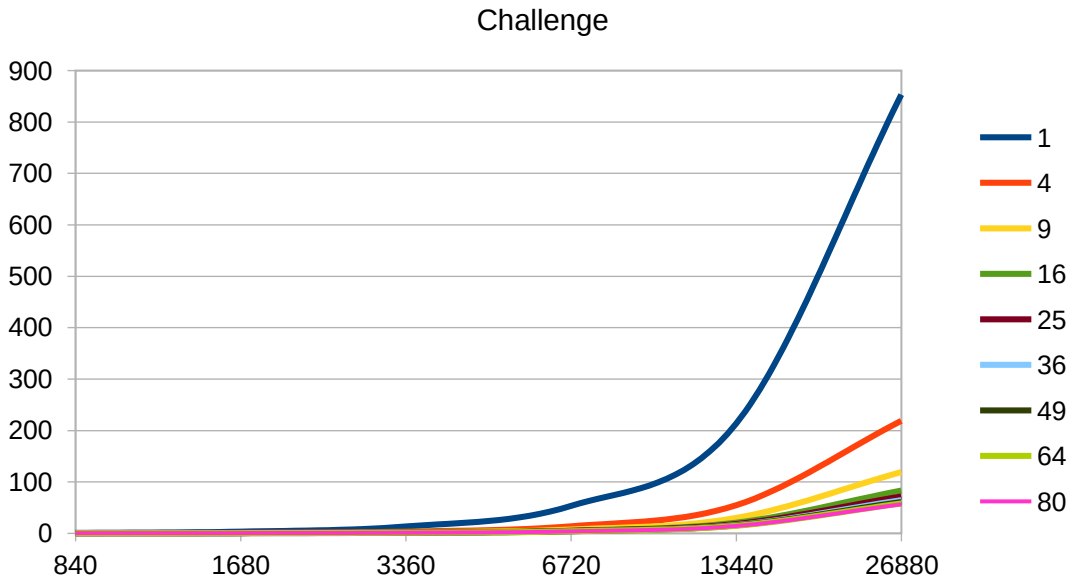
Βελτιστοποίηση MPI Κώδικα

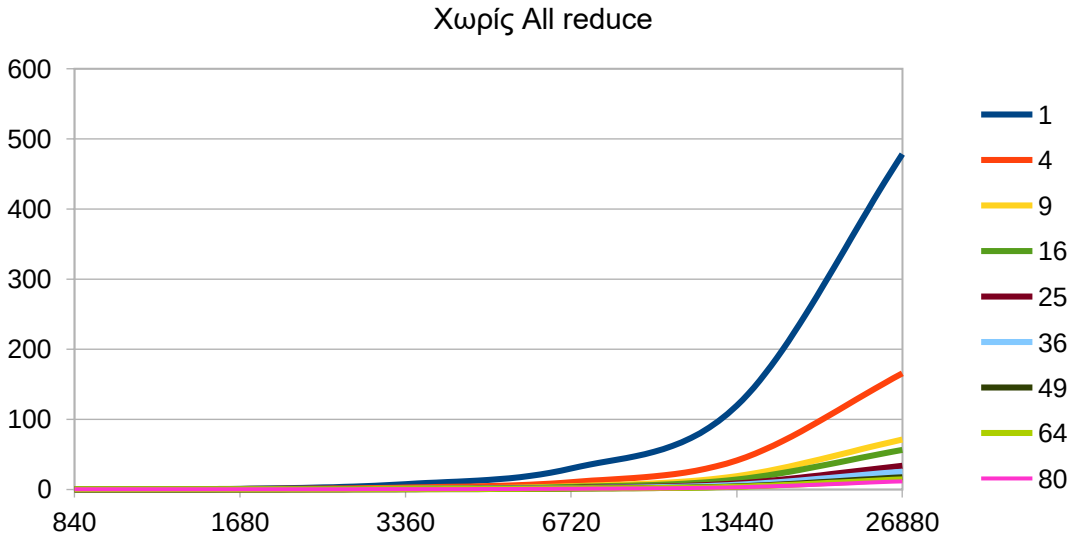
1. Οι διεργασίες έχουν επανατοποθετηθεί (reorder) αμέσως μετά τη δημιουργία του καρτεσιανού πλέγματος για τη μείωση κόστους επικοινωνίας.
2. Ο υπολογισμός των σταθερών γειτόνων «rightProc», «leftProc», «topProc» και «bottomProc» γίνεται μία φορά πριν την κεντρική επανάληψη.
3. Έχουμε ορίσει datatypes για τις σειρές και τις στήλες δεδομένων προς αποστολή για να αποφύγουμε τις αλλεπάλληλες απόπειρες επικοινωνία για κάθε ένα απ' τα στοιχεία που τη χρειάζονται. Επίσης δεν γίνεται αντιγραφή των δεδομένων, αλλά ανάθεση σε δείκτη στο πρώτο στοιχείο της δομής.
4. Στην κεντρική επανάληψη ξεκινάμε πρώτα με τα αιτήματα receive και έπειτα τα send για να είναι σε θέση οι διεργασίες να δεχθούν τα δεδομένα απευθείας.
5. Τα εσωτερικά στοιχεία ξεκινούν να υπολογίζονται πριν ολοκληρωθούν τα αιτήματα send/receive καθώς δε χρειάζονται τα δεδομένα που στέλνονται με μηνύματα.
6. Αφού ολοκληρωθούν οι υπολογισμοί των εσωτερικών στοιχείων και οι κατάλληλες wait που ακολουθούν, ξεκινούν να υπολογίζονται και τα εξωτερικά στοιχεία που χρειάζονται τα δεδομένα τα οποία έφτασαν στην άλω.

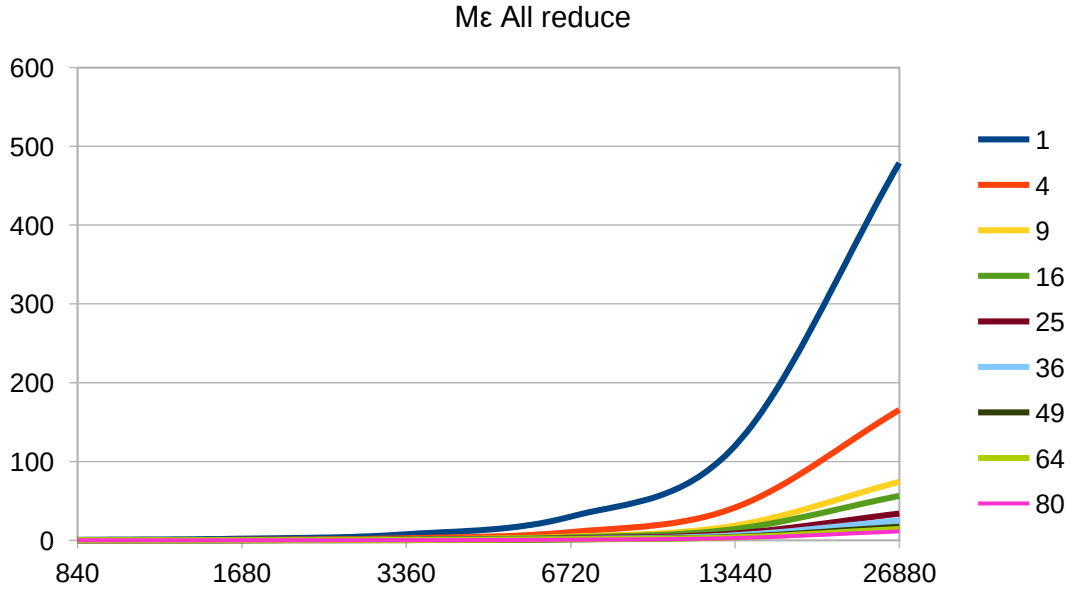
Μετρήσεις MPI κώδικα

- Έχουμε κάνει συνολικά 3 πλήρεις μελέτες καθαρού MPI κώδικα. Η πρώτη αφορά το δοσμένο πρόγραμμα του Challenge, ενώ οι άλλες δύο αφορούν τη δική μας υλοποίηση MPI, τη μία φορά με τη χρήση του «MPI_All_reduce» και την άλλη χωρίς.
- Το πρόγραμμά μας κατάφερε και στις δύο περιπτώσεις να ξεπεράσει το δοσμένο πρόγραμμα σε όλες τις μετρήσεις, ενώ παρατηρήσαμε μία ελαφρά μεγαλύτερη καθυστέρηση με τη χρήση του all reduce από όλες τις διεργασίες στην κεντρική επανάληψη σε σχέση με την απλή συγκέντρωση των αποτελεσμάτων απ' τη διεργασία 0 στο τέλος κάθε άλλης διεργασίας. Αυτό είναι λογικό μιας και με την all reduce όλες οι διεργασίες επικοινωνούν σε κάθε επανάληψη. Το επαληθεύσαμε και με τη βοήθεια των mpiP reports, αφού στην περίπτωση του all reduce, αυτό εισάγει την μεγαλύτερη καθυστέρηση, ενώ στην απλή υλοποίηση, η μεγαλύτερη τιμή καθυστέρησης προέρχεται απ' τις εντολές wait.
- Παρατηρήσαμε μία ασυνήθιστη συμπεριφορά του προγράμματός μας και με τις 2 υλοποιήσεις όταν χρησιμοποιήσαμε 64 διεργασίες στο πρόβλημα των 6720x6720 στοιχείων, ωστόσο δεν έχουμε σίγουρο συμπέρασμα για το λόγο που προκαλείται αυτό μιας και απ' τις μελέτες profiling τα δεδομένα έχουν μοιραστεί αρκετά ισορροπημένα μεταξύ των διεργασιών της τοπολογίας. Έπειτα η επιτάχυνση αυξάνεται και πάλι, στους ρυθμούς που αναμένεται για το πρόβλημα των 26880x26880
- Έχουμε κάνει μετρήσεις και για το πρόβλημα των 53760x53760, μόνο για 25, 36, 49, 64 και 80 διεργασίες ή 36, 64 και 80 νήματα. Για μικρότερο αριθμό διεργασιών/νημάτων η μνήμη δεν επαρκεί για να τρέξει το πρόγραμμα, επομένως δεν έχουμε συμπεριλάβει αυτές τις τιμές στα διαγράμματα, αλλά μόνο στους πίνακες μετρήσεων. Επίσης τα αποτελέσματα σε αυτό το μέγεθος βγαίνουν αρνητικά (πιθανό overflow;), αλλά οι χρόνοι είναι οι αναμενόμενοι.

Διαγράμματα χρόνου/διεργασιών

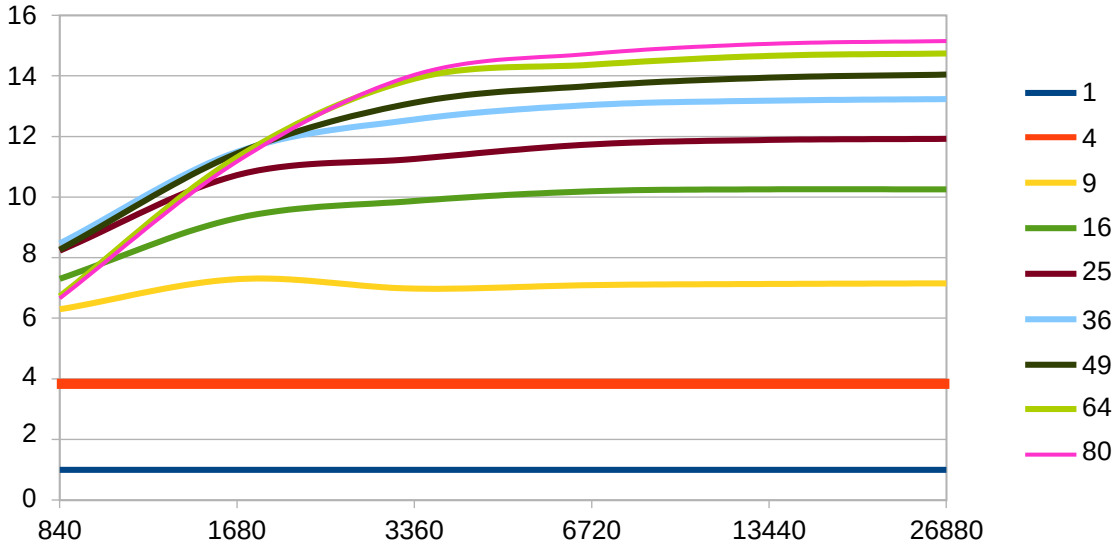




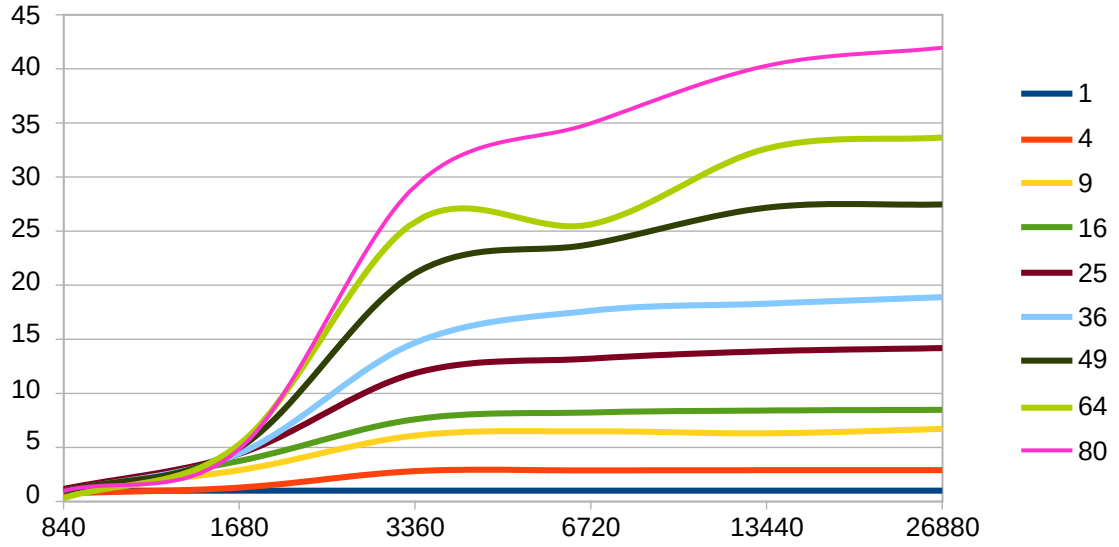


Διαγράμματα επιτάχυνσης/διεργασιών

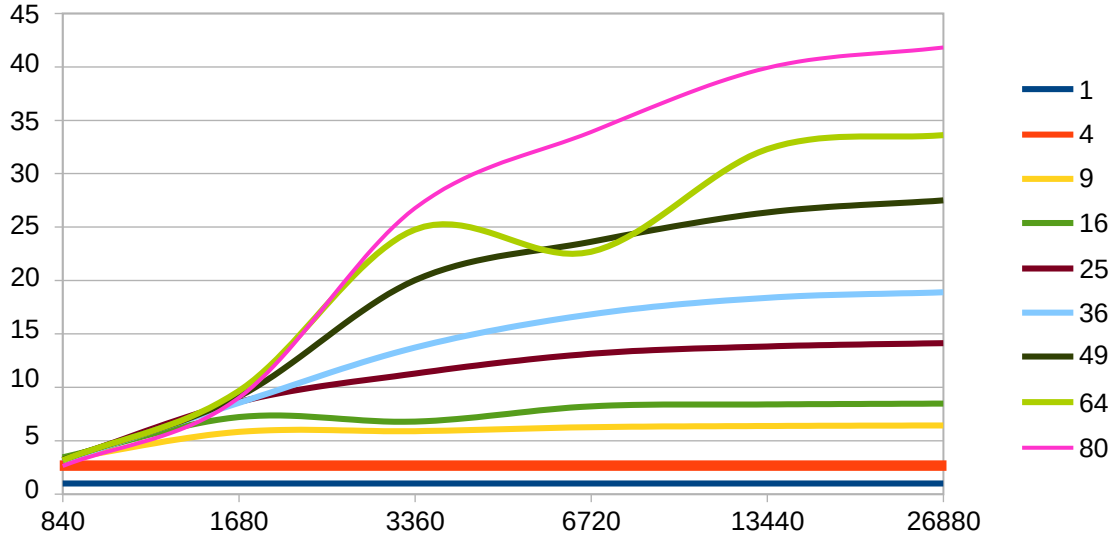
Challenge



Χωρίς All reduce

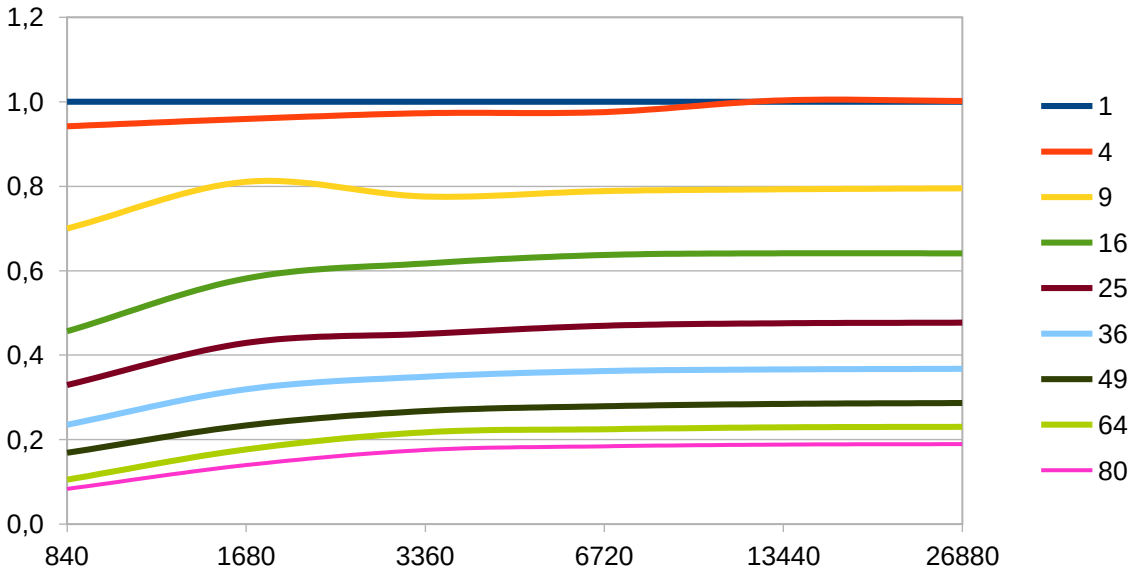


Διάγραμμα χρόνου/διεργασιών με All reduce

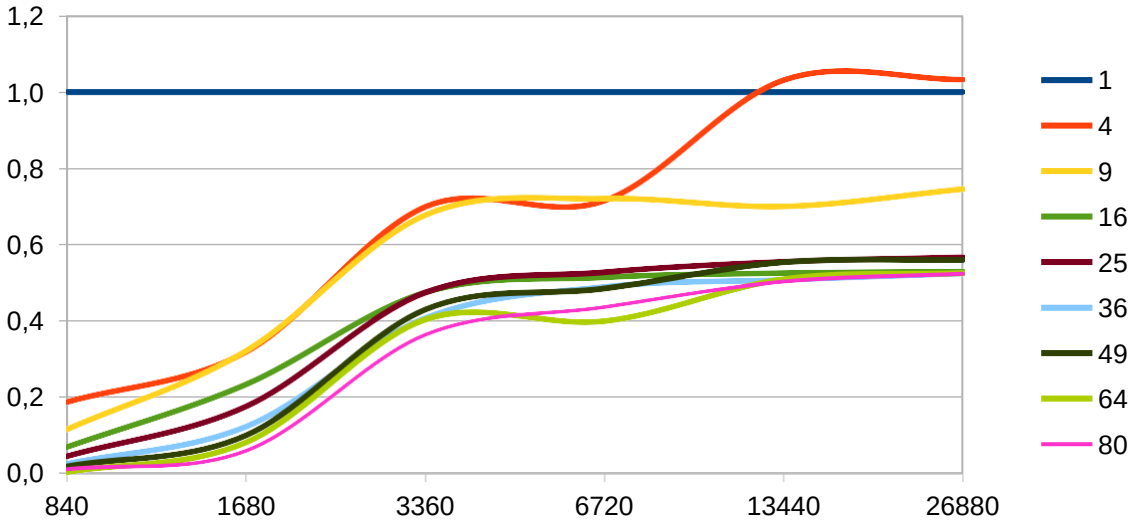


Διαγράμματα απόδοσης/διεργασιών

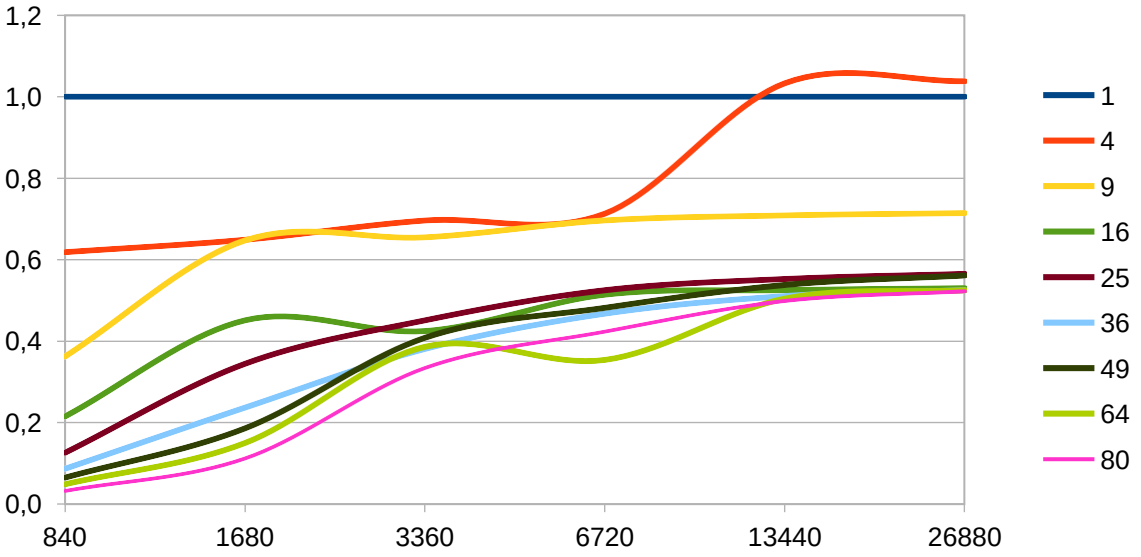
Challenge



Χωρίς All reduce



Με All reduce



- Μπορείτε να δείτε αναλυτικά του πίνακες χρόνων, επιτάχυνσης και απόδοσης στα αντίστοιχα excel αρχεία που περιλαμβάνονται σε κάθε έναν απ' τους υποφακέλους του φακέλου Measurements.

Σχόλια για μετρήσεις MPI:

Πέρα απ' το σημείο των 64 διεργασιών για πρόβλημα 6720, το πρόγραμμα δείχνει να κλιμακώνει όπως αναμένεται. Για μεγάλο αριθμό διεργασιών παρατηρούμε έως και 3 φορές μεγαλύτερη επιτάχυνση σε σχέση με το Challenge, ενώ σημειώνουμε ακόμη μία υπεργραμμική επιτάχυνση στις 4 διεργασίες για προβλήματα μεγέθους 13440x13440 και 26880x26880, το οποίο συμβαίνει και στο δοσμένο πρόγραμμα αλλά σε μικρότερο βαθμό. Επομένως σε αυτές τις μετρήσεις το efficiency είναι πάνω από 1.

Υβριδικό MPI + OpenMp

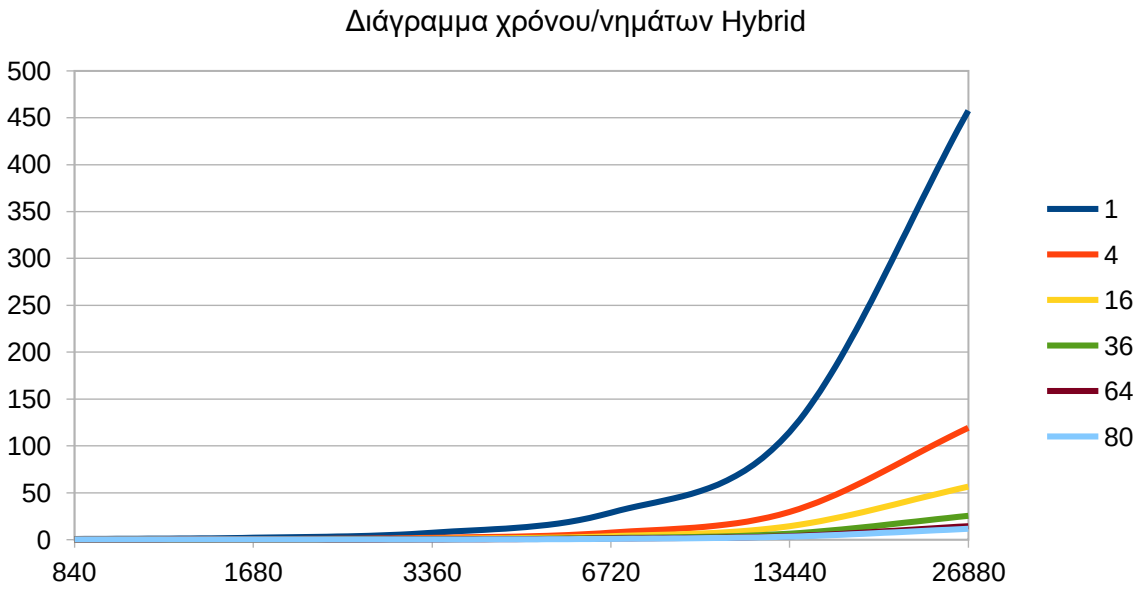
Για την υλοποίηση του υβριδικού κώδικα, εξελίξαμε τον κώδικα mpi μας (χωρίς All reduce), παραλληλοποιόντας όλες τις εσωτερικές for σε νήματα. Έπειτα από μετρήσεις για προβλήματα διάφορων μεγεθών καταλήξαμε στη χρήση 4 νημάτων ανά διεργασία, επειδή αυτή η υλοποίηση είχε τους καλύτερους χρόνους σε σχέση με 2 ή >4 νήματα(με μικρή διαφορά).

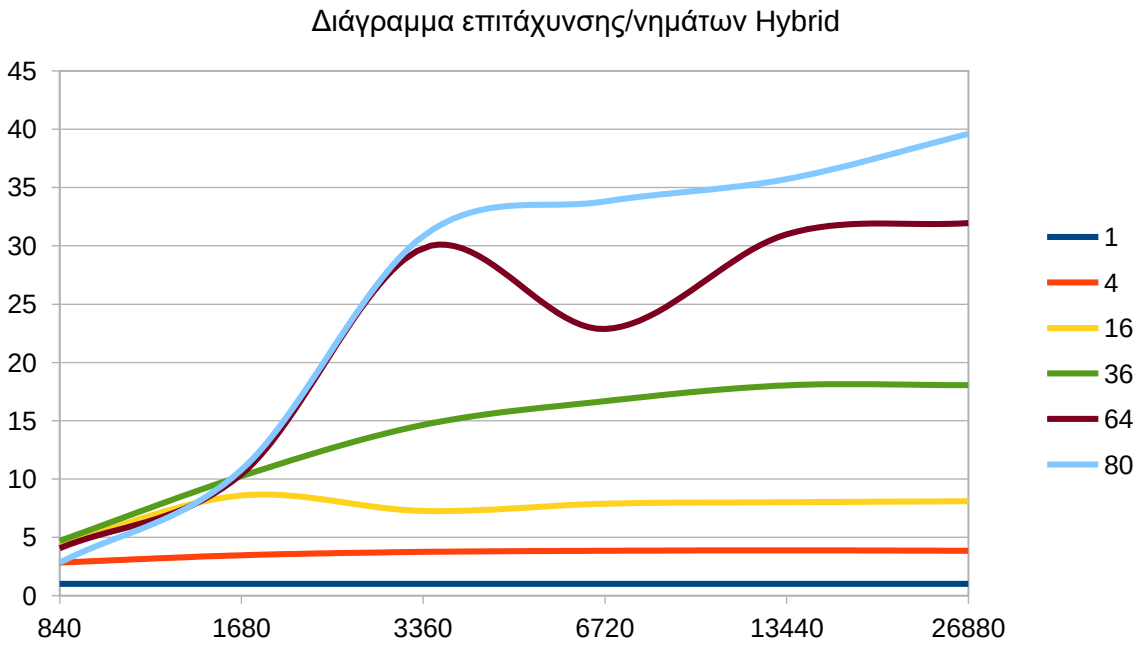
Σχόλια για μετρήσεις MPI & OpenMp:

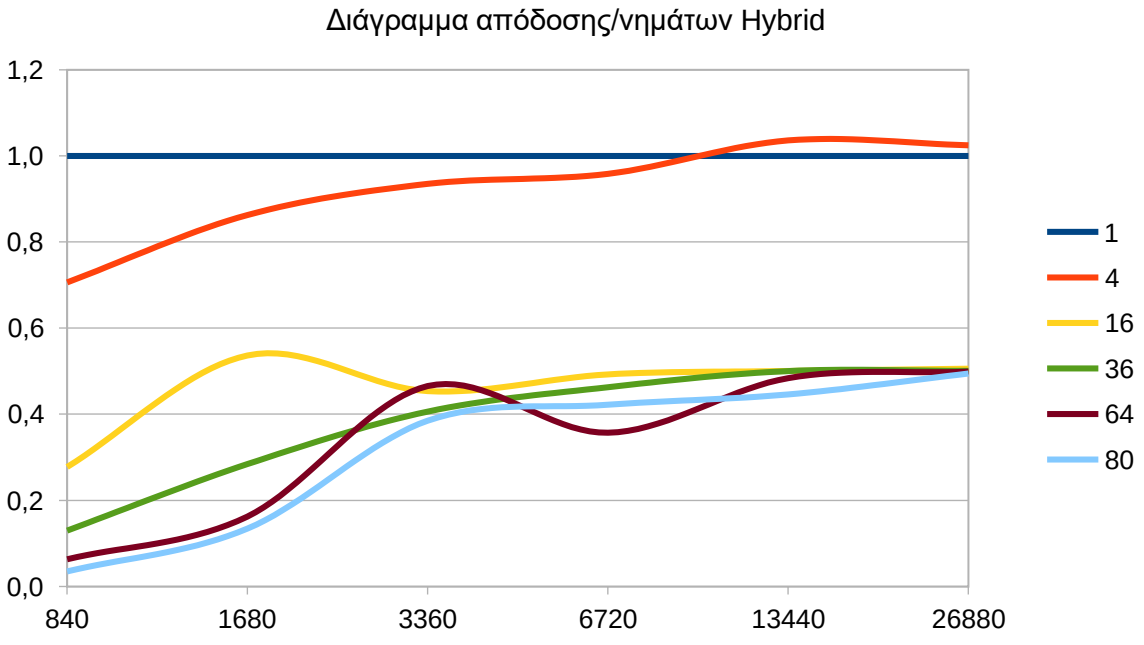
Οι χρόνοι είναι παραπλήσιοι με το σκέτο MPI όταν έχουμε μικρό αριθμό νημάτων, αλλά παρατηρούμε ότι όσο αυξάνεται η χρήση τους (με αποκορύφωμα τα 80 νήματα) τότε το υβριδικό πρόγραμμα έχει καλύτερη συμπεριφορά. Αυτό συμβαίνει διότι, σε μεγάλο αριθμό διεργασιών έχουμε μεγάλη ανάγκη για επικοινωνία των γειτόνων, ενώ με τη χρήση νημάτων η επικοινωνία αυτή μειώνεται, επειδή τα νήματα είναι ελαφριές διεργασίες, μοιράζονται την ίδια μνήμη και έτσι η επικοινωνία τους γίνεται εσωτερικά ως κομμάτι της ίδια διεργασίας. Για παράδειγμα με τη χρήση 4 νημάτων ανά διεργασία, στην περίπτωση των 80 νημάτων έχουμε μόνο 20 διαφορετικές διεργασίες να επικοινωνούν, σε αντίθεση με τις 80 του καθαρού MPI. Γι' αυτό το λόγο το όφελος του υβριδικού παρατηρείται όλο και περισσότερο για μεγάλο αριθμό νημάτων.

Παραθέτουμε παρακάτω το διάγραμμα χρόνου, επιτάχυνσης και απόδοσης για το υβριδικό πρόγραμμα, ενώ μπορείτε να δείτε αναλυτικά τους πίνακες αυτών των δεδομένων στο αρχείο excel που βρίσκεται στο φάκελο «Measurements/Hybrid_MPI_&_OpenMp»

Παρατήρηση: Και στην περίπτωση του υβριδικού προγράμματος βλέπουμε μία μη αναμενόμενη πτώση της απόδοσης για 64 νήματα στο πρόβλημα 6720x6720, αλλά και την υπεργραμμική επιτάχυνση για 4 νήματα σε προβλήματα μεγάλου μεγέθους.


Διάγραμμα χρόνου/νημάτων Hybrid
500
450
400
350
300
250
200
150
100
50
0
840
1680
3360
6720
13440
26880
1
4
16
36
64
80


Διάγραμμα επιτάχυνσης/νημάτων Hybrid
45
40
35
30
25
20
15
10
5
0
840
1680
3360
6720
13440
26880
1
4
16
36
64
80


Διάγραμμα απόδοσης/νημάτων Hybrid
1,2
1,0
0,8
0,6
0,4
0,2
0,0
840
1680
3360
6720
13440
26880
1
4
16
36
64
80

Συμπεράσματα

- Η βελτιστοποίηση του ακολουθιακού κώδικα, έπαιξε καθοριστικό ρόλο στην απόδοση και του παράλληλου προγράμματος, μιας και μόνο με τις αλλαγές που κάναμε σε αυτό το κομμάτι, το πρόβλημα λύνεται περίπου δύο φορές γρηγορότερα για το πρόβλημα 840x840 με μία διεργασία ή σχεδόν 3 φορές γρηγορότερα για το πρόβλημα 26880x26880 με χρήση 80 διεργασιών/νημάτων.
- Παρατηρούμε ότι το συγκεκριμένο πρόβλημα για τα μεγαλύτερα δυνατά μεγέθη των 13440 και 26880, παρουσιάζει υπεργραμμική επιτάχυνση για 4 διεργασίες/νήματα και επίσης έχει πολύ καλή απόδοση για 9 διεργασίες. Επομένως συμπεραίνουμε ότι το πρόβλημα μέχρι και το μέγιστο μέγεθος που μπορούμε να τρέξουμε στην ΑΡΓΩ, έχει την καλύτερη απόδοση όταν τα δεδομένα μοιράζονται σε 4 ή 9 ίσα μέρη. Άρα, αυτοί είναι οι αριθμοί διεργασιών/νημάτων που εξισορροπείται η καθυστέρηση λόγω επικοινωνίας και ο χρόνος που χρειάζεται κάθε υποπρόβλημα για να επιλυθεί ατομικά.
- Συγκρίνοντας το καθαρό MPI με το υβριδικό πρόγραμμα, είναι πιο αποδοτικό όταν είναι μικρότερος ο αριθμός των διεργασιών, ενώ το δεύτερο υπερισχύει για μεγαλύτερο αριθμό νημάτων. Η διαφορά, όμως, των χρόνων είναι αρκετά μικρή, επομένως και τα δύο είναι εξίσου αποδοτικά.