Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής & Τηλεπικοινωνιών
Παράλληλα Υπολογιστικά Συστήματα
Σεπτέμβριος 2019

Παραλληλοποίηση Αλγορίθμου Προσομοίωσης Μεταφοράς Θερμότητας

Γιώργος Κατσογιάννης-Μεϊμαράκης
sdi1400065@di.uoa.gr

Γιάννης Χήρας
sdi1400225@di.uoa.gr

Περιεχόμενα

1 Εισαγωγή

HEAT2D is based on a simplified two-dimensional heat equation domain decomposition. The initial temperature is computed to be high in the middle of the domain and zero at the boundaries. The boundaries are held at zero throughout the simulation. During the time-stepping, an array containing two domains is used; these domains alternate between old data and new data. At each time step, worker processes must exchange border data with neighbors, because a grid point's current temperature depends upon its previous time step value plus the values of the neighboring grid points.

Για κάθε τετράγωνο του πλέγματος η τιμή της θερμοκρασίας στο επόμενο time-step υπολογίζεται από την εξής εξίσωση:

$$\begin{aligned} u'[x][y] = u[x][y] \\ &+ c_x * (u[x+1][y] + u[x-1][y] - 2 * u[x][y]) \\ &+ c_y * (u[x][y+1] + u[x][y-1] - 2 * u[x][y]) \end{aligned}$$

Όπου $c_x = c_y = 0.1$ σταθερές.

2 Μεταγλώττιση & Εκτέλεση

2.1 Μεταγλώττιση

Τα προγράμματα μεταγλωττίζονται με τις παρακάτω εντολές και παράγουν ένα εκτελέσιμο το οποίο δίνεται προς εκτέλεση στην ουρά του συστήματος.

Για τη μεταγλώττιση προγραμμάτων που θα χρησιμοποιηθούν για μετρήσεις, δίνουμε στον complier το flag -O3 για τη βελτιστοποίηση του κώδικα σε χαμηλό επίπεδο.

- `mpi_heat2Dn.c`: Βελτιωμένος κώδικας MPI χωρίς σύγκλιση

```
mpicc -O3 mpi_heat2Dn.c -o mpi_heat2Dn.x
```

- `mpi_heat2Dn_conv.c`: Βελτιωμένος κώδικας MPI με σύγκλιση

```
mpicc -O3 mpi_heat2Dn_conv.c -o mpi_heat2Dn.x
```

- `mpi_heat2Dn_hybrid.c`: Βελτιωμένος υβριδικός κώδικας MPI + OpenMP με σύγκλιση

```
mpicc -O3 -fopenmp mpi_heat2Dn_hybrid.c -o mpi_heat2Dn.x
```

2.2 Εκτέλεση

Για την εκτέλεση του προγράμματος στο cluster του hellasgrid.gr χρησιμοποιούμε το script pbs_script.sh

```
qsub pbs_script.sh
```

3 Εργαλεία Ανάπτυξης Project

3.1 Version Control (Git/Github)

Για την καλύτερη διαχείρηση των εκδόσεων του κώδικα και των αλλαγών καθώς και για τη συνεργασία μεταξύ των μελών της ομάδας χρησιμοποιείται το πρόγραμμα git και η πλατφόρμα Github.

4 Οργάνωση Αρχείων & Φακέλων

Ο κώδικας για κάθε διαφορετική έκδοση που περιγράφουμε βρίσκεται και σε διαφορετικό αρχείο. Αυτό συνέβη κυριώς για δική μας διευκόλυνση στην εκτέλεση των δοκιμών.

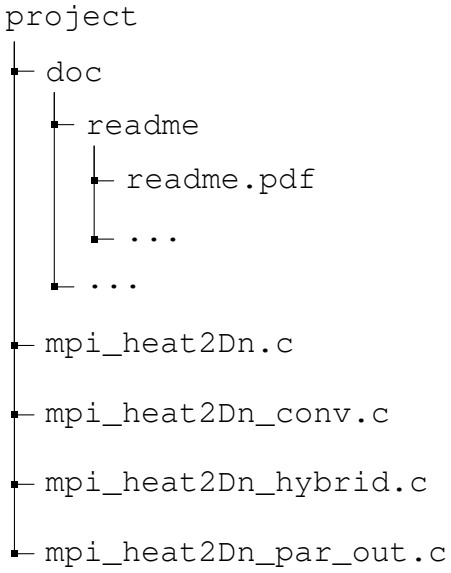

Figure 1: Κατάλογος Αρχείων

5 Βελτιωμένο Πρόγραμμα MPI

5.1 Ανάλυση Βελτιώσεων

5.1.1 Διαμοιρασμός σε 2 Διαστάσεις

Ο διαμοιρασμός του πίνακα γίνεται πλέον σε 2 διαστάσεις αντί μιας. Χρησιμοποιούμε το cartesian topology κομμάτι (βιβλιοθήκη?) του MPI για αυτό το διαμοιρασμό. Έτσι, κάθε διεργασία έχει πλέον εώς και 4 γείτονες, ενώ το taskid της αλλάζει ωστε να αντιπροσωπεύει τη θέση της στον καρτεσιανό διαχωρισμό.

5.1.2 Άλω Εξωτερικών Σημείων

Το μέγεθος πίνακα κάθε διεργασίας έχει αυξηθεί κατά 2 σε κάθε κατεύθυνση (1 στήλη στην αρχή και στο τέλος, 1 σειρά στην αρχή και στο τέλος) σχηματίζοντας άλω (halo points). Εκεί αποθηκεύονται οι γειτονικές τιμές που απαιτούνται απο τα εξωτερικά σημεία για τον υπολογισμό της νέας τιμής τους. Η ανταλλαγή αυτών των πληροφοριών δεν γίνεται μεμονωμένα, αλλά με αποστολή/λήψη ολόκληρων στηλών και σειρών ανά φορά, για αποφυγή καθυστέρησης (latency) στην επικοινωνία.

5.1.3 Workflow Επικοινωνιών - Υπολογισμός

Το κυρίως τμήμα της επανάληψης έχει αλλάξει δραστικά δομή.

Χρησιμοποιούμε non-blocking επικοινωνία (Isend) για να στείλουμε τις τιμές των εξωτερικών σημείων στους ανάλογους γείτονές της διεργασίας, ενώ αντίστοιχα λαμβάνονται (Irecv) χωρίς blocking οι τιμές των halo points. Μέχρι να ολοκληρωθεί η επικοινωνία, η διεργασία υπολογίζει τις εσωτερικές τιμές του grid της, οι οποίες βασίζονται σε δεδομένα που ήδη έχει.

Έπειτα περιμένει να ολοκληρωθεί η λήψη των halo points, με 4 Wait για τα ανάλογα requests. Αφού τα λάβει, η διεργασία υπολογίζει την νέα τιμή των εξωτερικών στοιχείων της. Τέλος, αφού περιμένει βεβαίωση ολοκλήρωσης της αποστολής (4 ακόμα Wait), αλλάζει τον "νέο" πίνακα σε "παλιό", αλλάζοντας απλά την μεταβλητή που φέρει τον ενεργό πίνακα.

5.1.4 Πρώτα Receive - Μετά Send

Η νέα σειρά της λήψης (1η) και της αποστολής (2η), που επιτρέπεται λόγω του non-blocking communication, βεβαιώνει το ότι οι διεργασίες θα είναι έτοιμες για λήψη, όταν γίνει η αποστολή,μειώνοντας την καθυστέρηση.

5.1.5 Datatypes Σειρών - Στηλών

Χρησιμοποιούμε 2 δομές (MPI_Type_vector) για να περιγράψουμε τις στήλες και τις σειρές. Έτσι η ανταλλαγή τους γίνεται άμεσα και αποφεύγεται αντιγραφή τιμών, ή χρήση buffer, για τη λήψη και την αποστολή.

Συγκεκριμένα, η δομή της σειράς (MPI_ROW) έχει columns στοιχεία, δηλαδή όσες στήλες έχει ο πίνακας της διεργασίας. Κάθε στοιχείο είναι ένας MPI_FLOAT, και απέχει 1 στοιχείο απο το επόμενο.

Αντίστοιχα, η δομή της στήλης, έχει rows στοιχεία, δηλαδή όσες σειρές έχει ο πίνακας της διεργασίας. Κάθε στοιχείο είναι ένας MPI_FLOAT, και απέχει columns+2 στοιχεία απο το επόμενο.

5.1.6 Υπολογισμός Γειτόνων

Ο υπολογισμός των γειτόνων κάθε διεργασίας γίνεται μόνο μια φορά στην αρχή, μέσω συναρτήσεων της καρτεσιανής τοπολογίας. Οι γείτονες φυσικά είναι σταθεροί κατά τη διάρκεια μιας εκτέλεσης.

5.1.7 Αποφυγή Ifs για Επικοινωνίες Γειτόνων

Ορίζουμε τα ranks για τους ανύπαρκτους γείτονες κάθε διεργασίας ως MPI_PROC_NULL, ώστε να αποφεύγονται περιττοί έλεγχοι (if, case).

Οι τιμές στις αντίστοιχες θέσεις του halo αρχικοποιούνται σε 0, και δεν αλλάζουν.

5.1.8 Γείτονες σε ίδιο κόμβο

Οι γειτονικές διεργασίες τοποθετούνται όσο είναι δυνατόν στους ίδιους κόμβους, για να ελαττώνεται η καθυστέρηση του κόστους επικοινωνίας. Αυτό επιτυγχάνεται αυτόματα μέσω των διεργασίων του cartesian topology. Ελέγχουμε την επιτυχία του με το MPI_Get_processor_name.

5.1.9 Persistent Communication

Καθώς οι γείτονες είναι σταθεροί, εφαρμόζουμε persistent communication.

Δημιουργούμε 2 πίνακες send request (s_array), και 2 receive request (r_array), έναν για κάθε instance του πίνακα (1ο - 2ο). Κάθε θέση αφορά σε έναν γείτονα, με MPI_Send_init/MPI_Recv_init στους αντίστοιχους πίνακες. Κάθε φορά χρησιμοποιείται το request του "τωρινού" πίνακα. Μέσα στην επανάληψη, απλά καλείται με MPI_Start η ανάλογη επικοινωνιακή συνάρτηση, με τους ήδη αποθηκευμένους παραμέτρους. Έτσι αποφεύγουμε τον επαναυπολογισμό τους.

5.1.10 Δυναμικοί Πίνακες

Αποφεύγουμε την αντιγραφή τιμών μεταξύ των πινάκων, ανταλλάζοντας απλά την τιμή της μεταβλητής iz, που υποδεικνύει τον τωρινό πίνακα, απο 0 σε 1 και αντίστροφα.

5.2 Χρόνοι

Στο παρακάτω πίνακα φαίνονται οι μετρήσεις χρόνων που πήραμε τρέχοντας το πρόγραμμα mpi_heat2Dn στο cluster του hellasgrid.

Nodes	**Tasks**	80×64	160×128	320×256	640×512	1280×1024	2560×2048	5120×4096
1	1	0.003476	0.013983	0.116129	0.464293	1.907506	7.492858	30.003448
1	4	0.001333	0.004159	0.014871	0.117549	0.472859	1.935913	7.563494
2	16	0.118759	0.153217	0.013015	0.012097	0.098547	0.053514	2.033478
8	64	0.281022	0.414862	0.061213	0.301717	0.326839	0.316296	0.527302
16	128	0.041406	0.131406	0.433416	0.321896	0.487963	0.296483	0.274705
20	160	0.071904	0.089528	0.596713	0.487695	0.521694	0.273331	0.205910

Table 1: Χρόνοι Εκτέλεσης mpi_heat2Dn

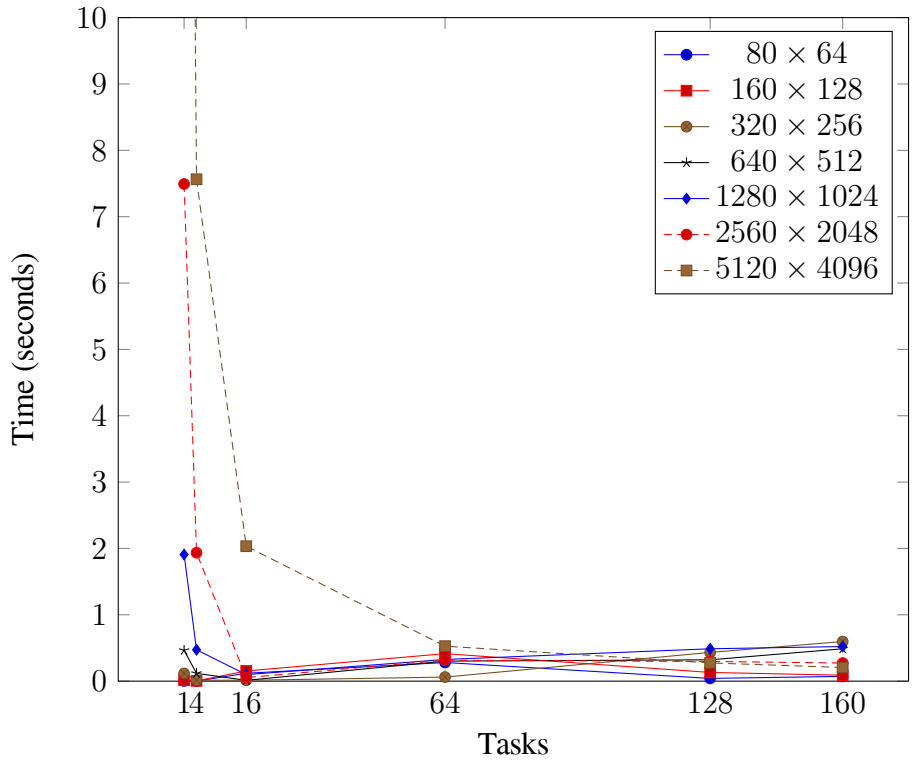

Figure 2: Χρόνοι Εκτέλεσης mpi_heat2Dn προς πλήθος Tasks

5.3 Speed-up

Για τους χρόνους που μετρήσαμε προκύπτουν τα εξής speed-up.

Nodes	**Tasks**	80 × 64	160 × 128	320 × 256	640 × 512	1280 × 1024	2560 × 2048	5120 × 4096
1	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	4	2.61	3.36	7.81	3.95	4.03	3.87	3.97
2	16	0.03	0.09	8.92	38.38	19.36	140.02	14.75
8	64	0.01	0.03	1.90	1.54	5.84	23.69	56.90
16	128	0.08	0.11	0.27	1.44	3.91	25.27	109.22
20	160	0.05	0.16	0.19	0.95	3.66	27.41	145.71

Table 2: Speed-up για το mpi_heat2Dn

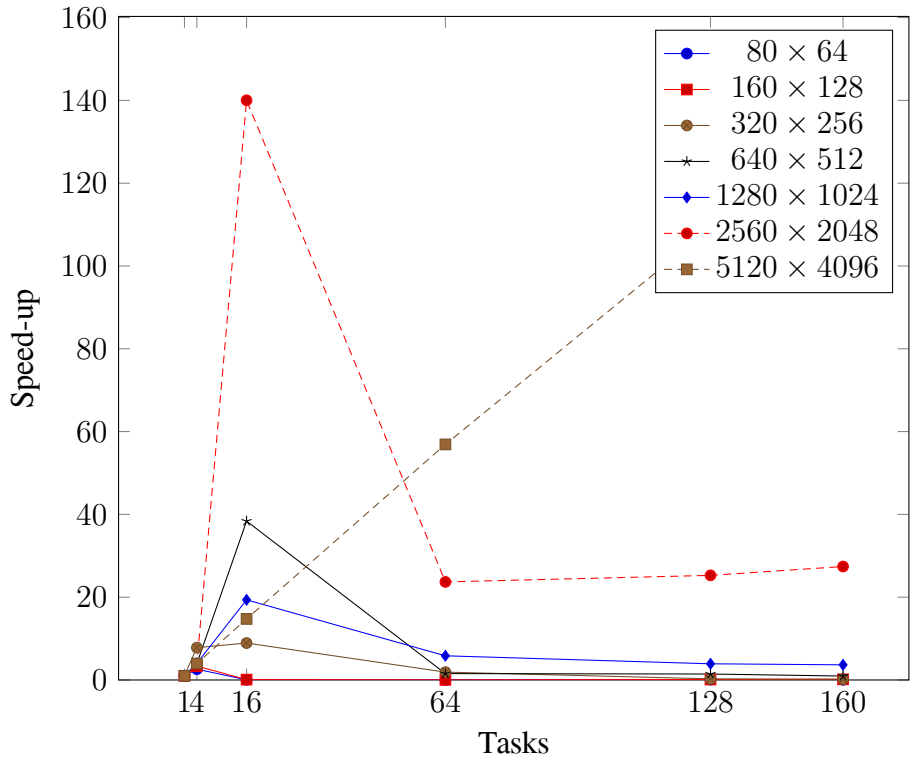

Figure 3: Speed-up του mpi_heat2Dn προς πλήθος Tasks

5.4 Efficiency

Αντίστοιχα για τους συγκεκριμένους χρόνους προκύπτουν οι εξής τιμές efficiency.

Nodes	**Tasks**	80 × 64	160 × 128	320 × 256	640 × 512	1280 × 1024	2560 × 2048	5120 × 4096
1	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	4	2.61	3.36	7.81	3.95	4.03	3.87	3.97
2	16	0.01	0.05	4.46	19.19	9.68	70.01	7.38
8	64	0.00	0.00	0.24	0.19	0.73	2.96	7.11
16	128	0.01	0.01	0.02	0.09	0.24	1.58	6.83
20	160	0.00	0.01	0.01	0.05	0.18	1.37	7.29

Table 3: Efficiency για το mpi_heat2Dn

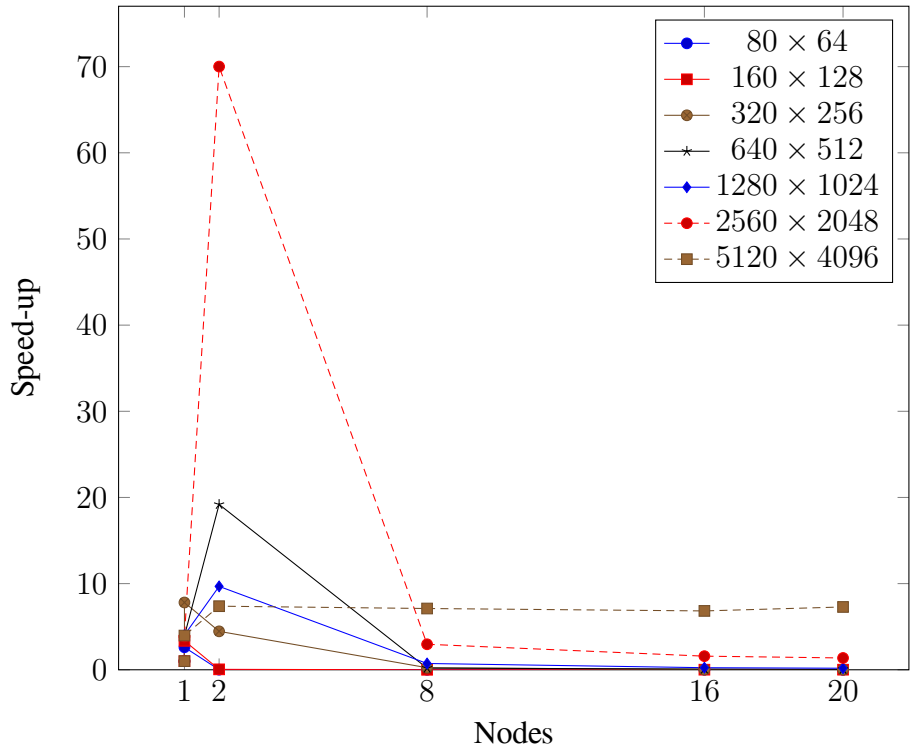

Figure 4: Efficiency του mpi_heat2Dn προς πλήθος Nodes

6 Βελτιωμένο Πρόγραμμα MPI με Σύγκλιση

6.1 Γενικά

Ο έλεγχος σύγκλισης τιμών συμβαίνει ανά CONV_PERIOD βήματα της επανάληψης. Στα κατάλληλα βήματα, η συνάρτηση υπολογισμού των νέων τιμών update αλλάζει, και προστίθεται σε αυτή ο έλεγχος για διαφορά των παλιών/νέων τιμών. Έτσι αποφεύγεται ενα περιττό διπλό for/iteration του πίνακα αρχικά για τον υπολογισμό των νέων τιμών και στη συνέχεια για τον έλεγχο αυτόν.

Στο τέλος του αναλόγου βήματος, γίνεται το reduce που υπολογίζει τη συνολική σύγκλιση των διεργασιών.

Στο συγκεκριμένο project, καθώς θέλαμε όλα τα βήματα των ελέγχων να είναι ίσα, δεν εκμεταλλευόμαστε την πληροφορία αυτή, αλλά έχουμε υπογραμμίσει με σχόλιο το σημείο που η διακοπή του αλγορίθμου θα γινόταν.

6.2 Χρόνοι

Παρακάτω βρίσκονται οι χρόνοι εκτέλεσης του προγράμματος αφού προσθέσαμε έλεγχο σύγκλισης όπως περιγράφουμε παραπάνω.

Nodes	**Tasks**	80×64	160×128	320×256	640×512	1280×1024	2560×2048	5120×4096
1	1	0.003599	0.014306	0.116803	1.912541	3.587840	14.385803	30.130099
1	4	0.001452	0.004271	0.015229	0.468824	0.469402	1.943977	7.594688
2	16	0.012620	0.802080	0.105832	1.995662	2.004883	2.232759	0.023318
8	64	0.704997	0.398689	0.191372	0.203051	0.533432	0.564690	0.045449
16	128	0.567120	0.533837	0.325748	0.316250	0.384510	0.475910	0.071355
20	160	0.400491	0.356482	0.354197	0.342627	0.318850	0.214963	0.105473

Table 4: Χρόνοι Εκτέλεσης mpi_heat2Dn_conv

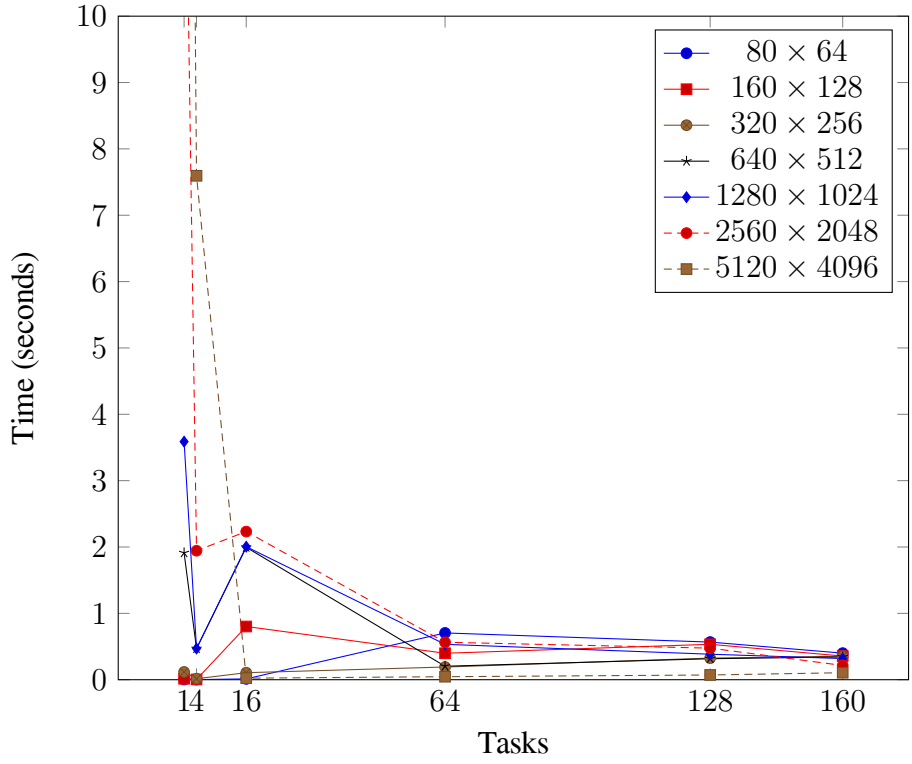

Figure 5: Χρόνοι Εκτέλεσης mpi_heat2Dn_conv προς πλήθος Tasks

6.3 Speed-up

Αντίστοιχα για αυτούς τους χρόνους εκτέλεσης υπολογίζουμε το speed-up που προκύπτει:

Nodes	**Tasks**	80×64	160×128	320×256	640×512	1280×1024	2560×2048	5120×4096
1	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	4	2.48	3.35	7.67	4.08	7.64	7.40	3.97
2	16	0.29	0.02	1.10	0.96	1.79	6.44	1292.14
8	64	0.01	0.04	0.61	9.42	6.73	25.48	662.94
16	128	0.01	0.03	0.36	6.05	9.33	30.23	422.26
20	160	0.01	0.04	0.33	5.58	11.25	66.92	285.67

Table 5: Speed-up για το mpi_heat2Dn_conv

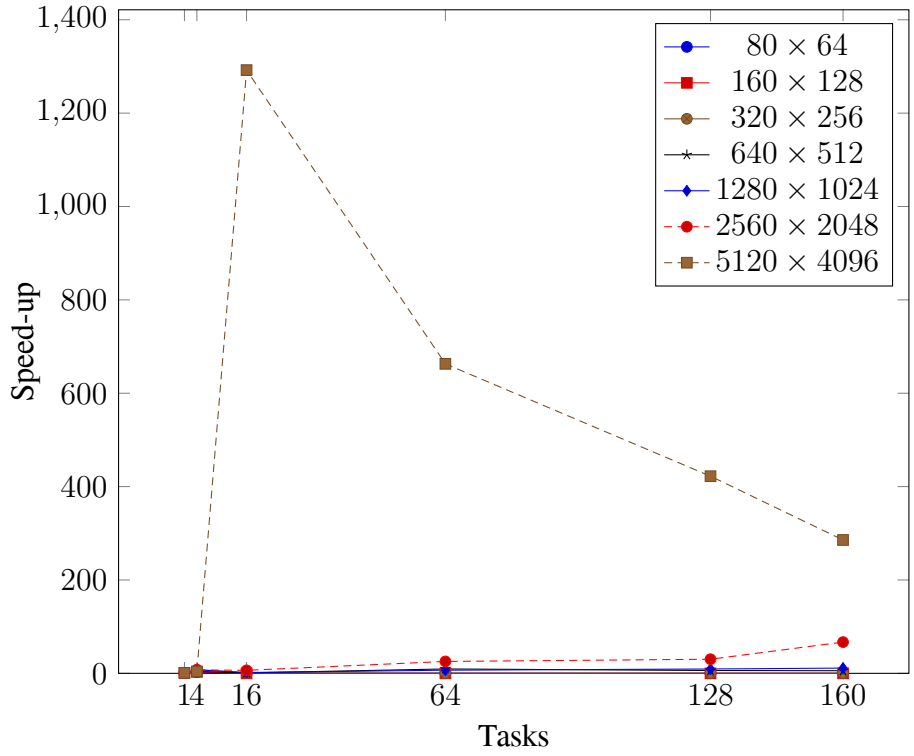

Figure 6: Speed-up του mpi_heat2Dn_conv προς πλήθος Tasks

6.4 Efficiency

Επίσης για τους χρόνους εκτέλεσης παρατηρούνται τα εξής efficincies ανα περίπτωση.

Nodes	**Tasks**	80×64	160×128	320×256	640×512	1280×1024	2560×2048	5120×4096
1	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	4	2.48	3.35	7.67	4.08	7.64	7.40	3.97
2	16	0.14	0.01	0.55	0.48	0.89	3.22	646.07
8	64	0.00	0.00	0.08	1.18	0.84	3.18	82.87
16	128	0.00	0.00	0.02	0.38	0.58	1.89	26.39
20	160	0.00	0.00	0.02	0.28	0.56	3.35	14.28

Table 6: Efficiency για το mpi_heat2Dn_conv

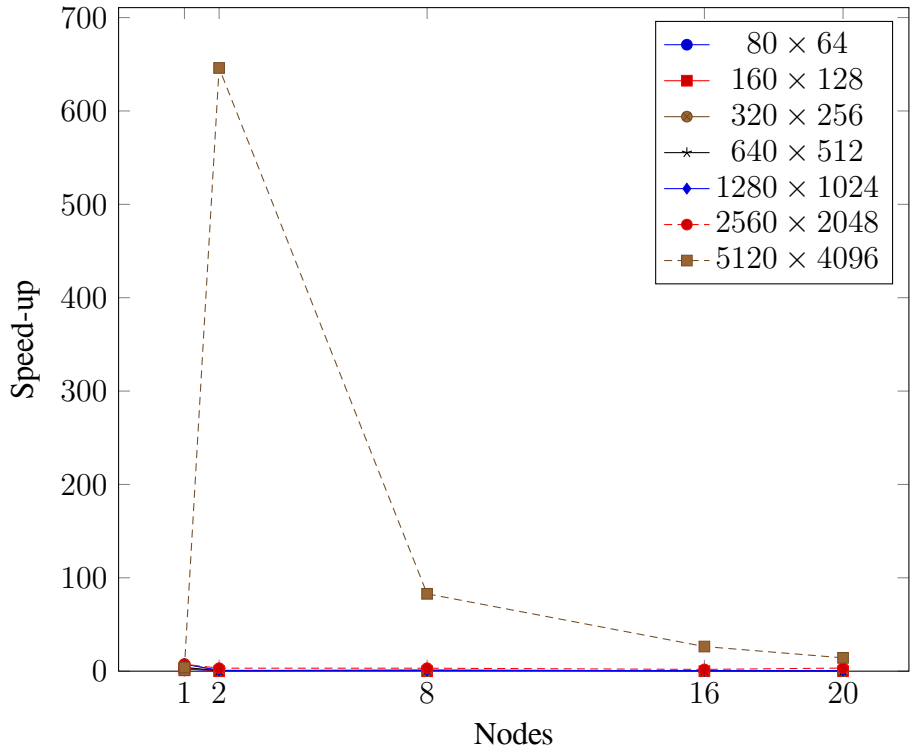

Figure 7: Efficiency του mpi_heat2Dn_conv προς πλήθος Nodes

7 Βελτιωμένο Υβριδικό Πρόγραμμα MPI & OpenMP με Σύγκλιση

7.1 Γενικά

Στη συνέχεια δημιουργήσαμε ένα υβριδικό πρόγραμμα συνδιάζοντας MPI και OpenMP. Με τη χρήση του OpenMp δημιουργήσαμε threads τα οποία μοιράζονται το φόρτο των for loops, προσπαθώντας έτσι να βελτιώσουμε την απόδοση του προγράμματος.

7.2 Βελτιώσεις

7.2.1 Δημιουργία Threads

Για να είναι όσο το δυνατόν πιο αποδοτικό το εν λόγω πρόγραμμα, η δημιουργία των threads αντί να γίνεται κάθε φορά που φτάνουμε στο σημείο των υπολογισμών, γίνεται εκτός της κύριας επανάληψης. Όταν φτάσουμε στο σημείο των υπολογισμών, κάνουμε schedule στα threads την δουλειά που πρέπει να γίνει.

7.2.2 Έλεγχος σύγλισης

Όπως και στο προηγούμενο πρόγραμμα, ο έλεγχος σύγκλισης γίνεται ταυτόχρονα με τον υπολογισμό των νέων τιμών έτσι ώστε να αποφύγουμε να κάνουμε τις διπλάσιες επαναλήψεις πάνω στον πίνακα.

7.3 Χρόνοι

Στη συνέχεια παραθέτουμε τους χρόνους εκτέλεσης του υβριδικού προγράμματος για διάφορους αριθμούς nodes, processes per node και threads.

Nodes	**PPN**	**Threads**	80×64	160×128	320×256	640×512	1280×1024	2560×2048	5120×4096
1	1	4	0.017807	0.076955	0.849709	6.237978	18.845583	94.873767	419.162658
1	2	2	0.010130	0.054823	0.302259	4.974396	5.028509	76.641657	346.727030
1	2	4	0.010141	0.042359	0.573377	5.160128	13.141330	77.193930	346.580322
2	2	8	0.041168	0.046581	0.307811	4.093556	1.288874	4.958769	343.488488
2	4	4	0.048170	0.054576	0.117058	0.605236	1.287806	4.941937	37.707930
2	8	2	0.050319	0.113991	0.077582	0.397800	2.373664	9.473987	19.348335
2	8	4	0.045401	0.053395	0.090122	0.350299	13.168804	75.159566	19.255832
8	32	2	0.0451497	1.458103	1.566198	2.001983	2.741039	3.373491	4.997810
8	32	4	0.0571659	1.054486	1.977853	2.744596	5.358996	9.114651	13.694557
8	16	4	6.149755	0.994236	2.094377	3.649588	5.001397	7.811678	9.335702
8	8	8	0.084391	0.197436	0.374159	0.569843	0.742369	1.422739	4.036750
16	64	2	2.547896	2.784369	2.974612	2.897413	3.014716	3.364197	3.941678
16	64	4	1.557396	2.317885	2.569332	3.468102	3.671539	5.039125	8.997672
16	32	4	0.686951	1.247556	2.648337	5.276559	4.168227	5.066149	7.330571
16	16	8	0.0579463	1.722579	2.165993	3.943687	4.379551	4.089271	9.325899

Table 7: Χρόνοι Εκτέλεσης mpi_heat2Dn_hybrid

7.4 Speed-up

Στον παρακάτω πίνακα βλέπουμε το speed-up που προκύπτει για κάθε περίπτωση.

Nodes	**PPN**	**Threads**	80×64	160×128	320×256	640×512	1280×1024	2560×2048	5120×4096
1	1	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	2	2	1.76	1.40	2.81	1.25	3.75	1.24	1.21
1	2	4	1.76	1.82	1.48	1.21	1.43	1.23	1.21
2	2	8	0.43	1.65	2.76	1.52	14.62	19.13	1.22
2	4	4	0.37	1.41	7.26	10.31	14.63	19.20	11.12
2	8	2	0.35	0.68	10.95	15.68	7.94	10.01	21.66
2	8	4	0.39	1.44	9.43	17.81	1.43	1.26	21.77
8	32	2	0.39	0.05	0.54	3.12	6.88	28.12	83.87
8	32	4	0.31	0.07	0.43	2.27	3.52	10.41	30.61
8	16	4	0.00	0.08	0.41	1.71	3.77	12.15	44.90
8	8	8	0.21	0.39	2.27	10.95	25.39	66.68	103.84
16	64	2	0.01	0.03	0.29	2.15	6.25	28.20	106.34
16	64	4	0.01	0.03	0.33	1.80	5.13	18.83	46.59
16	32	4	0.03	0.06	0.32	1.18	4.52	18.73	57.18
16	16	8	0.31	0.04	0.39	1.58	4.30	23.20	44.95

Table 8: Speed-up για το mpi_heat2Dn_hybrid

7.5 Efficiency

Τέλος, στο επόμενο πίνακα παραθέτουμε το efficiency που προκύπτει για κάθε περίπτωση.

Nodes	**PPN**	**Threads**	80×64	160×128	320×256	640×512	1280×1024	2560×2048	5120×4096
1	1	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	2	2	1.76	1.40	2.81	1.25	3.75	1.24	1.21
1	2	4	1.76	1.82	1.48	1.21	1.43	1.23	1.21
2	2	8	0.22	0.83	1.38	0.76	7.31	9.57	0.61
2	4	4	0.18	0.71	3.63	5.15	7.32	9.60	5.56
2	8	2	0.18	0.34	5.48	7.84	3.97	5.01	10.83
2	8	4	0.20	0.72	4.71	8.90	0.72	0.63	10.88
8	32	2	0.05	0.01	0.07	0.39	0.86	3.52	10.48
8	32	4	0.04	0.01	0.05	0.28	0.44	1.30	3.83
8	16	4	0.00	0.01	0.05	0.21	0.47	1.52	5.61
8	8	8	0.03	0.05	0.28	1.37	3.17	8.34	12.98
16	64	2	0.00	0.00	0.02	0.13	0.39	1.76	6.65
16	64	4	0.00	0.00	0.02	0.11	0.32	1.18	2.91
16	32	4	0.00	0.00	0.02	0.07	0.28	1.17	3.57
16	16	8	0.02	0.00	0.02	0.10	0.27	1.45	2.81

Table 9: Efficiency για το mpi_heat2Dn_hybrid

8 Parallel Output

Σχεδιάσαμε ένα σύστημα Parallel Output των τιμών του πίνακα κάθε διεργασίας σε ένα αρχείο.

Η διεργασία υπολογίζει βάσει των καρτεσιανών συντεταγμένων της το σημείο του αρχείου στο οποίο πρέπει να γράψει, και τυπώνει το κομμάτι του τελικού πίνακά της εκεί (δίχως φυσικά τα halo points).

Συγκεκριμένα, ο τρόπος που υπολογίζεται η απόσταση (σε bytes απο την αρχή του αρχείου) στην οποία πρέπει να γράψει μια διεργασία, είναι η εξής :

Αρχικά υπολογίζεται το μέγεθος σε bytes του output μίας διεργασίας (output_size), καθώς και μίας σειράς (row_size). Γνωρίζουμε ότι όλες οι διεργασίες σε πιο ”πάνω” καρτεσιανή σειρά του πίνακα θα έχουν γράψει όλα τα δεδομένα τους πριν απο τη συγκεκριμένη διεργασία. Έτσι, σίγουρα προχωράμε κατά output_size * coord[0] * cart_dims[1].

Έπειτα, καθώς κάθε σειρά μιας διεργασίας πρέπει να γράφεται μετά την αντίστοιχη του αριστερού, και πριν την αντίστοιχη του δεξίου της γείτονα, προχωράμε επαναληπτικά κατα row_size * coord[1]. Εκεί τυπώνουμε την σειρά με κενά ανάμεσα στους αριθμούς. Αν η διεργασία δεν έχει δεξί γείτονα, τον τελευταίο αριθμό κάθε σειρά της ακολουθεί ενα newline αντί κενού.

Δεν μπορέσαμε να ελέγξουμε διεξοδικά το σύστημα αυτό, λόγω φόρτου των μηχανημάτων του hellasgrid.