ΕΡΓΑΣΙΑ ΠΑΡΑΛΛΗΛΩΝ ΣΥΣΤΗΜΑΤΩΝ 2018-2019

Προσομοίωση μεταφοράς θερμότητας

Μέλη:

Γιαννακίδης Ιωάννης 1115 2015 00025
Ταβουλάρης Φώτης Έκτορας 1115 2015 00154

Π Ε Ρ Ι Ε Χ Ο Μ Ε Ν Α

1. ΕΙΣΑΓΩΓΗ

Η παρούσα εργασία, εκπονείται στα πλαίσια του μαθήματος “Παράλληλα Υπολογιστικά Συστήματα” και αποτελεί μια προσομοίωση μεταφοράς θερμότητας σε επιφάνεια με παράλληλη επεξεργασία σε επίπεδο επεξεργαστή (CPU) με χρήση MPI καθώς και OpenMp και σε επίπεδο κάρτας γραφικών (GPU) με την χρήση CUDA.

2. ΣΧΕΔΙΑΣΜΟΣ ΔΙΑΜΟΙΡΑΣΜΟΥ

ΔΕΔΟΜΕΝΩΝ

Ο διαμοιρασμός των δεδομένων γίνεται σε block ίσων μεγεθών. Κάθε task δεσμεύει το μέγεθος που του αντιστοιχεί (ανάλογα με το πλήθος των tasks) συν δύο rows και δύο columns. Ο λόγος που δεσμεύουμε την έξτρα μνήμη είναι ώστε να έχουμε μια γραμμή γύρω από τα δεδομένα μας με όλες τις τιμές 0. Με αυτό τον τρόπο αν κάποιο task δεν έχει γείτονα, δηλαδή είναι στα άκρα, αυτή η πλευρά θα μείνει μηδέν, ενώ αν έχει γείτονα θα αντιγράψει εκεί τις τιμές που θα του σταλθούν ώστε να μπορεί μετά να κάνει τους υπολογισμούς του. Με αυτόν τον τρόπο, αρχικά

δεν χρειαζόμαστε κάποιο master task για την αρχική αρχικοποίηση και διαμοιρασμό δεδομένων καθώς και για την αναμονή της ολοκλήρωσης των υπόλοιπων task, κερδίζοντας έτσι ένας επιπλέον ‘εργάτη’.

Κάθε task υπολογίζει τους γείτονες του (LEFT,RIGHT,UP,DOWN) αν υπάρχουν, ώστε να ξέρει με ποιους θα έχει επικοινωνία πριν αρχίσει την κεντρική επανάληψη και θέτει τους κενούς γείτονες σε MPI_PROC_NULL ώστε να μην χρειάζεται έλεγχος μέσα στην επανάληψη. Δημιουργούμε ένα νέο Communicator και δύο νέα Datatypes για την πιο δομημένη και γρήγορη επικοινωνία ενώ ταυτόχρονα αποφεύγεται η ανάθεση τιμών και η χρήση buffer.

3. ΣΧΕΔΙΑΣΜΟΣ MPI ΚΩΔΙΚΑ

I. Σχεδιασμός MPI

Για την βελτιστοποίηση της απόδοσης του αλγορίθμου σε MPI ακολουθήσαμε τα παρακάτω βήματα. Η δέσμευση της μνήμης γίνεται μια φορά (δέσμευση μεγέθους (NXPROB + 2) * (NYPROB + 2) ώστε οι θέσεις που ισαπέχουν στο block, να ισαπέχουν και στη μνήμη (πχ. Το U[0][1] απέχει από το U[1][1] όσο και η θέσεις μνήμης τους). Αν η δέσμευση της μνήμης γινόταν όπως συνηθίζεται, δηλαδή ανά row ή ανά column, το λειτουργικό θα είχε αποθηκεύσει ανάμεσα σε κάθε malloc και ένας δείκτη στην μνήμη που δεσμεύαμε και έτσι θα έσπαγε η συνοχή.

Χρησιμοποιούμε την MPI_Cart_create για την δημιουργία ενός νέου communicator και περνάμε τις τοπογραφικές πληροφορίες που αντιστοιχούν καθώς ξέρουμε ότι η επικοινωνία γίνεται μόνο κάθετα με τον ακριβώς πάνω και κάτω γείτονα και οριζόντια με τον ακριβώς δεξιά και αριστερά. Μετά δημιουργούμε δύο MPI types, το MPI_row και το MPI_column. Το πρώτο είναι MPI_FLOATS σε διπλανές θέσεις μνήμης, ενώ το δεύτερο είναι MPI_FLOAT που απέχουν μεταξύ τους μέγεθος ίσο με ένα MPI_row. Λόγω αυτών των Datatypes δεν χρειάζεται η χρήση buffer ή ανάθεσης για την αποστολή των columns, που σε άλλη περίπτωση θα δημιουργούσαν πρόβλημα καθώς δεν είναι σε συνεχόμενες θέσεις μνήμης.

Στην συνέχεια θέτουμε όποιο γείτονα δεν υπάρχει σε NULL ώστε να μήν χρειάζεται ο έλεγχος μέσα στην επανάληψη και καλούμε MPI_Barrier για τον communicator που δημιουργήσαμε ώστε να συγχρονιστούν όλες οι διαδικασίες.

Η επανάληψη περιέχει τα Isend και Irecv για να επιτευχθεί nonblocking επικοινωνία ακολουθώντας την ίδια μεθοδολογία με το αρχικό πρόβλημα όσον αφορά τα TAGS. Στην θέση του communicator request έχουμε χρησιμοποιήσει διαφορετική μεταβλητή για κάθε isend/irecv ώστε να τα ελέγχουμε όλα στα wait που ακολουθούν. Για τον υπολογισμό του heat transfer χρησιμοποιείται το ίδια

συνάρτηση update όπως στο αρχικό πρόγραμμα με την διαφορά ότι το μέγεθος του σημείου εφαρμογής είναι μεταβλητό.

II. Σχεδιασμός OpenMp

Για την υλοποίηση ενός αποδοτικού υβριδικού κώδικα ακολουθήσαμε μια πολύ απλή προσέγγιση. Παραλληλοποιήσαμε με χρήση ενός διπλού for τον υπολογισμό των νέων στοιχείων στην συνάρτηση update_hv. Επίσης, η αρχικοποίηση των thread γίνεται εκτός του κεντρικού loop προκειμένου να γίνεται μόνο 1 φορά για την απαραίτητη αποφυγή της καθυστέρησης. Για αυτόν τον λόγο βάλαμε και κάποιους ελέγχους στο κεντρικό loop προκειμένου τα MPI calls να πραγματοποιούνται μόνο από 1 thread. Για αυτόν τον σκοπό χρησιμοποιήσαμε το #pragma omp master έναντι του #pragma omp single καθώς παρατηρήσαμε πως το πρόγραμμα είχε πιο γρήγορη εκτέλεση με αυτόν τον τρόπο

III. Σχεδιασμός Cuda

Για τον αλγόριθμο σε Cuda ξεκινάμε δεσμεύοντας μνήμη ίση με NYPROB * NYPROB στην μνήμη και το διπλό μέγεθος στην μνήμη της κάρτας γραφικών για να έχουμε δύο πίνακες cuda_u0, cuda_u1 ώστε να αντιστοιχούν στον πίνακα u του MPI (cuda_u0 = u[0], cuda_u1 = u[1]). Στην συνέχεια αρχικοποιούμε με τυχαίες τιμές τον πίνακα του host και τον αντιγράφουμε στους δύο πίνακες του device.

Σε αντίθεση με το MPI πρόγραμμα, η επανάληψη κινείται με βήμα δύο και σε κάθε επανάληψη η συνάρτηση update καλείται δύο φορές, κάθε φορά με διαφορετικό πίνακα στην θέση του παλιού.

Μετά την επανάληψη, η τιμή του device πίνακα αντιγράφεται στον host και γίνονται τα απαραίτητα free τόσο στο device όσο και στο host.

4. ΜΕΤΡΗΣΕΙΣ ΧΡΟΝΟΥ ΕΚΤΕΛΕΣΗΣ

- Μετρήσεις χρόνου στο rsb.hellasgrid.gr
- Αριθμός steps = 500
- Χρόνος σε δευτερόλεπτα (seconds)

a. MPI χωρίς έλεγχο σύγκλισης

Μέγεθος block/ # διεργασιών	80 X 64	160 X 128	320 X 256	640 X 1024
1	2.345321	0.126730	0.629308	4.601321
4	1.153319	0.019327	0.132387	1.158545
16	0.155229	0.167029	0.051123	0.430522
64	0.163025	0.428888	0.031283	0.520341
128	0.111348	0.454728	0.023192	0.609472
160	0.144532	0.487549	0.021389	0.729341

SPEEDUP-S(n)=ts/tp για μέγεθος block 320 X 256

4	16	64	128	160
4.753	12.309	2.011	27.134	2.874

EFFICIENCY-E=S(n)/n

4	16	64	128	160
1.188	0.769	0.031	0.211	0.017

b. Mpi με έλεγχο σύγκλισης κάθε 50 επαναλήψεις και ευαισθησια 0.8

Μέγεθος block/ # διεργασιών	80 X 64	160 X 128	320 X 256	640 X 1024
1	2.324567	0.068337	0.569291	4.618549
4	1.153781	0.019384	0.072598	1.221268
16	0.186839	0.071700	0.053895	0.377538
64	0.177631	0.083245	0.202797	0.427634
128	0.182121	0.091212	0.223287	0.646396
160	0.198721	0.092345	0.248946	0.759362

SPEEDUP-S(n)=ts/tp για μέγεθος block 320 X 256

4	16	64	128	160
7.841	10.562	2.807	2.549	2.286

EFFICIENCY-E=S(n)/n

4	16	64	128	160
1.96	0.66	0.043	0.018	0.014

c. Mpi με έλεγχο σύγκλισης κάθε 50 επαναλήψεις και ευαισθησία 0.8 + OpenMp

I. 2 threads

Μέγεθος block/ # διεργασιών	80 X 64	160 X 128	320 X 256	640 X 1024
2(1X2)	0.041771	0.072709	0.572657	4.595384
8(1X8)	1.596233	1.132451	0.080557	3.046082
32(4X8)	0.124532	3.565432	0.251196	0.647923
64(8X8)	0.144321	4.725141	0.314891	0.439207
80(10X8)	0.185329	4.931456	0.335572	0.398701

SPEEDUP-S(n)=ts/tp για μέγεθος block 320 X 256

8	32	64	80
7.108	2.279	1.818	1.706

EFFICIENCY-E=S(n)/n

8	32	64	80
0.888	0.071	0.028	0.021

II. 4 threads

Μέγεθος block/ # διεργασιών	80 X 64	160 X 128	320 X 256	640 X 1024
2(1X2)	0.048878	0.146110	1.151275	22.123208
8(1X8)	1.487523	0.122345	1.679799	4.341209
32(4X8)	4.678220	0.156781	1.324562	1.982012
64(8X8)	3.324356	0.213284	1.897621	1.679201
80(10X8)	3.876586	0.723191	1.546723	1.723094

SPEEDUP-S(n)=ts/tp για μέγεθος block 320 X 256

8	32	64	80
0.685	0.869	0.606	0.744

EFFICIENCY-E=S(n)/n

8	32	64	80
0.085	0.027	0.009	0.009

III. 4 threads

Μέγεθος block/ # διεργασιών	80 X 64	160 X 128	320 X 256	640 X 1024
1(1X1)	0.078565	0.282517	2.310712	32.206787
4(1X4)	2.965197	1.566768	2.008491	6.660996
16(2X8)	0.105643	4.560100	4.620940	2.117934
32(4X8)	0.118577	4.598753	0.156917	0.657233
40(5X8)	0.129087	4.615481	0.188245	0.589852

SPEEDUP-S(n)=ts/tp για μέγεθος block 320 X 256

4	16	32	40
1.15	0.5	14.725	12.275

EFFICIENCY-E=S(n)/n

4	16	32	40
0.287	0.031	0.46	0.306

IV. 8 threads

Μέγεθος block/ # διεργασιών	80 X 64	160 X 128	320 X 256	640 X 1024
2(1X2)	0.830139	0.292989	2.292329	20.710122
8(1X8)	1.234564	0.102315	0.175964	1.764318
16(2X8)	4.636223	0.093176	0.230887	2.366111
20(4X5)	4.537104	0.323046	0.208639	1.993307

SPEEDUP-S(n)=ts/tp για μέγεθος block 320 X 256

8	16	20
13.027	9.928	10.987

EFFICIENCY-E=S(n)/n

8	16	20
1.628	0.62	0.549

5. ΣΥΓΚΡΙΣΗ ΑΡΧΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Η σύγκριση του αρχικού προγράμματος με το δικό μας MPI πρόγραμμα έχει αισθητές διαφορές.Αρχικά,από πλευράς χρόνου εκτέλεσης,στις μετρήσεις που πραγματοποιήσαμε σε δικό μας μηχάνημα στους πίνακες που ακολουθούν στην ενότητα 6 φαίνεται άμεσα η επιτάχυνση που υπάρχει στο MPI πρόγραμμα έναντι του αρχικού.

Οι υπόλοιπες διαφορές βρίσκονται σε επίπεδο κώδικα.Όπως αναλύθηκαν και στις ενότητες 2 και 3,μία αρχική μεγάλη διαφοροποίηση είναι ο διαχωρισμός των δεδομένων σε 1 block για κάθε task έναντι ενός μεγάλου πίνακα που διαμοιράζεται μεταξύ των task στο αρχικό πρόγραμμα καθώς και η δυναμική δέσμευση-αποδέσμευση του κάθε block. Επιπλεόν, η μη χρήση master task στο mpi πρόγραμμα παρουσιάζει μία σημαντική διαφορά καθώς δεν χρειάζεται να περιμένουμε την ολοκλήρωση όλων των task πριν προχωρήσουμε.

Στο επίπεδο του κεντρικού loop παρατηρούνται βέβαια οι περισσότερες αλλαγές-βελτιστοποιήσεις.Αρχικά,μέσω επικάλυψης της επικοινωνίας με υπολογισμούς καθώς και με την δίαταξη των send-receive. Επιπλεόν, αποφυγή περιττών υπολογισμών μέσα στο κεντρικό loop που επαναλαμβάνονται συνεχώς.Η χρήση datatypes έναντι αντιγραφής τιμών συμβάλλει σημαντικά έναντι του αρχικού προγράμματος για την αποστολή και λήψη των δεδομένων.Ακόμα,για την αποφυγή περιττών ελέγχων-υπολογισμών η χρήση MPI_PROC_NULL όταν δεν υπάρχει κάποιος γείτονας.

6. ΕΠΕΚΤΑΣΕΙΣ-ΔΙΚΕΣ ΜΑΣ ΜΕΤΡΗΣΕΙΣ

- Μετρήσεις σε laptop με επεξεργαστή τον Intel Core i7-6500U
- Αριθμός steps = 1000

a. Mpi χωρίς έλεγχο σύγκλισης

Μέγεθος block/ # διεργασιών	20 X 20	40 X 40	100 X 100	200 X 200	500 X 500

4	0.057	0.060	0.227	0.343	2.631
9	0.095	0.134	0.280	0.484	2.996
25	0.310	0.310	0.235	0.663	2.736
36	0.350	0.386	0.366	1.522	4.685

b. Mpi με έλεγχο σύγκλισης κάθε 50 επαναλήψεις με 0.8 ευαισθησία

Μέγεθος block/ # διεργασιών	20 X 20	40 X 40	100 X 100	200 X 200	500 X 500
4	0.018	0.021	0.089	0.380	2.637
9	0.059	0.059	0.151	0.855	2.756
16	0.092	0.142	0.182	0.497	2.724
25	0.207	0.216	0.311	1.06	2.432
36	0.295	1.07	0.434	1.04	5.440

c. Αρχικό πρόγραμμα

Μέγεθος πινακα/ # διεργασιών	20 X 20	40 X 40	200 X 200
4	0.397	0.333	1.138
9	0.504	0.683	1.558
16	0.758	0.994	2.367

7. ΕΠΙΛΟΓΟΣ

Με την χρήση παράλληλων δομών και με τις πολυάριθμες δοκιμές που εκτελέσαμε σε αυτές μπορούμε να κατανοήσουμε την χρησιμότητα τους ειδικά σε προβλήματα μεγαλύτερου μεγέθους. Με τα ανωτέρα μπορούμε να καταλήξουμε πως η παραλληλία είναι ένα σημαντικό εργαλείο για να έχουμε speed-up στην ταχύτητα αλλά και στην εγκυρότητα πολλών προγραμματιστικών προγραμμάτων που συναντάμε

8. ΠΕΡΙΕΧΟΜΕΝΑ ΕΡΓΑΣΙΑΣ

- README
- Makefile σε κάθε φάκελο
- Φάκελος cuda με το αρχείο *cuda_heat2Dn.cu*
- Φάκελος mpi με το αρχείο *mpi_heat2Dn.c*
- Φάκελος openmp με το αρχείο *mpi_omp_heat2Dn.c*

9. ΠΗΓΕΣ-ΒΙΒΛΙΟΓΡΑΦΙΑ

- Σημειώσεις μαθήματος “Παράλληλα Υπολογιστικά Συστήματα” - Γιάννης Κοτρώνης
- Παρουσιάσεις Nvidia
- http://devblogs.nvidia.com/parallelforall/cuda-pro-tip-occupancy-api-simplifies-launch-configuration/