Εργασία στα Παράλληλα Συστήματα

"Σχεδιασμός, Ανάπτυξη και Αξιολόγηση Παραλλήλων Προγραμμάτων σε MPI, Υβριδικό Mpi+OpenMp, Cuda που υλοποιούν την Jacobi with successive over-relaxation"

Περιεχόμενα

Ι. Εισαγωγή

Το αντικείμενο της εργασίας είναι η μελέτη και ανάπτυξη παράλληλων προγραμμάτων που υλοποιούν την μεθοδο jacobi (με τη παραλλαγή successive over-relaxation) για να λύσει αριθμητικά την εξίσωση Poisson:

$$(\nabla^2 - \alpha)u = \frac{d^2}{dx^2}u + \frac{d^2}{dy^2}u - \alpha u = f$$

Αρχικά έγινε βελτιστοποίηση του ακολουθιακού προγράμματος σε μια καλύτερη μορφή για παράλληλο προγραμματισμό με λιγότερες αναθέσεις και inline συναρτήσεις. Στην συνέχεια, παραλληλοποιήθηκε το πρόγραμμα χρησιμοποιώντας MPI, υβριδικό MPI+openMP και CUDA με 1 και 2 gpus. Μελετήσαμε με μετρήσεις και συγκρίσεις την κλιμάκωση του προγράμματος και στις τρεις αυτές υλοποιήσεις. Παρακάτω παρουσιάζονται αναλυτικότερα οι μορφοποιήσεις, οι μετρήσεις και οι συγκρίσεις που προαναφέρθηκαν καθώς και τα profiling που προκύπτουν.

Έχουν υλοποιηθεί όλα τα ζητούμενα της εργασίας. Και τα δύο μέλη της ομάδας (argo238, argo266) έχουμε δώσει τα δικαιώματα Read-Write-eXecute και οι κατάλογοι έχουν ακριβώς τα ίδια περιεχόμενα. Ο κατάλογος ProjectSubmission περιέχει τους υποκαταλόγους Sequential, ParallelMPI, HybridMPI, CUDA με απλά makefiles και τα PBSscripts όπως ζητούνται στην εκφώνηση. Στον φάκελο CUDA υπάρχουν 2 υποφάκελοι, ένας για την υλοποίηση με 1 GPU και ένας με την υλοποίηση για 2 GPUs.

II. Ακολουθιακό Πρόγραμμα. Βελτιστοποίηση, προετοιμασία για παραλληλοποίηση.

Όπως ήδη αναφέρθηκε για την βελτιστοποίηση του ακολουθιακού μειώθηκαν οι αναθέσεις και έγιναν inline οι συναρτήσεις. Αναλυτικότερα μερικές από τις αλλαγές που έγιναν για την μείωση των αναθέσεων είναι ότι πλέον οι μεταβλητές cx , cy ,cc υπολογίζονται πριν την while στην main συνάρτηση και δίνονται σαν ορίσματα στην συνάρτηση one_jacobi_iteration αποφεύγοντας έτσι πάρα πολλές αναθέσεις και υπολογισμούς που προηγουμένως γίνονταν σε κάθε επανάληψη. Inline έγιναν οι συναρτήσεις one_jacobi_iteration και checkSolution. Ακόμα παρατηρήθηκε οτι η μείωση μερικών αναθέσεων όπως η fx τείνουν να δημιουργούν μεγαλύτερη καθυστέρηση καθώς σε κάθε επανάληψη αντί να υπολογιστούν μόνο μια φορά τώρα με τον τρόπο αυτό υπολογίζονται 4 φορές, πράγμα που οδηγεί σε αρκετά μεγαλύτερους χρόνους εκτέλεσης του προγράμματος. Ωστόσο, πλέον τα fX και fY δεν υπολογίζονται κάθε φορά ως συνάρτηση του x και y αλλά σε κάθε loop προστίθεται το deltaX και deltaY μειώνοντας τους υπολογισμούς.Τέλος, στον υπολογισμό του updateval έγινε μια παραγοντοποίηση η οποία επίσης μειώνει τους υπολογισμούς:

f = -alpha*(1.0-fX*fX)*(1.0-fY*fY) - 2.0*(1.0-fX*fX) - 2.0*(1.0-fY*fY);

→

((fX*fX-1.0)*(alpha*(1.0-fY*fY) + 2.0) - 2.0*(1.0 - fY*fY))

Παρατηρούμε ότι οι χρόνοι του βελτιστοποιημένου ακολουθιακού είναι αισθητά ταχύτεροι από το πρόγραμμα του challenge (και το αρχικό πρόγραμμα).

Size of matrix/Διεργασίες	1
840x840	0,7951
1680x1680	3,1722
3360x3360	12,6812
6720x6720	50,7138
13440x13440	213,3130
26880x26880	853,3510

III. Σχεδιασμός και βελτίωση MPI Παραλληλοποίησης.

Για τον σχεδιασμό της MPI παραλληλοποίησης τον σημαντικότερο ρόλο έπαιξε το 2 dimensions cartesian topology δηλαδή η δημιουργία δισδιάστατων πινάκων για την επικοινωνία των process. Ο τρόπος αυτός διευκολύνει κατά πολύ την εύρεση των γειτόνων καθώς αναζητούνται με βάση συντεταγμένων μέσω της MPI_Cartshift και ειναι άμεση η πρόσβαση σε αυτούς. Μέσω της τοπολογίας εφαρμόζεται η μεθοδολογία Foster που αποσκοπεί σε καλύτερο διαμοιρασμό των δεδομένων. Για την αποστολή και παραλαβή των άλω δημιουργήθηκαν datatypes για στήλες και σειρές. Η επικοινωνία γίνεται μέσω MPI_ISend MPI_IRecv ώστε να μην μπλοκάρουν οι διεργασίες την στιγμή της κλήσης και χρησιμοποιούμε κατάλληλα waits όπου χρειάζεται. Πρώτα γίνονται τα receive και μετά τα send έτσι ώστε τα receive να είναι έτοιμα να δεχθούν δεδομένα. Σε αυτό το κομμάτι κάναμε επικάλυψη επικοινωνίας και γενικά ακολουθήσαμε όλες τις οδηγίες στο Β και στο Γ εκτός από το σημείο 9 που αναφέρεται στο persistent communication, το οποίο με τον τρόπο που το υλοποιήσαμε είδαμε ότι δεν βοηθούσε στον χρόνο οπότε δεν το κρατήσαμε. Ο αρχικός πίνακας σπάει σε υποπίνακες με μέγεθος ανάλογο των process και υπολογίζονται τα error και οι νέες τιμές σε αυτούς και συγκεντρώνονται μέσω Allreduce. Για το checkSolution κάθε διεργασία υπολογίζει το δικό της σφάλμα με βάση το δικό της u_old, xStart και yStart και έπειτα με mpi_reduce υπολογίζεται το συνολικό σφάλμα. Στην checkSolution δεν δόθηκε ιδιαίτερη βαρύτητα στην παραλληλοποίηση της και γενικά σε όλη την έκταση της εργασίας την χρησιμοποιούμε κυρίως για επαλήθευση των αποτελεσμάτων. Τα σημεία τα οποία φαίνονται ανάποδα σε σχέση με το ακολουθιακό στον κώδικα του jacobi έχουν γίνει ώστε οι περιπτώσεις στις οποίες ο αριθμός των process δημιουργεί υποπίνακες με διαφορετικές διαστάσεις σε συνδυασμό με την τοπολογία να οδηγούν σε ένα σωστό αποτέλεσμα.

VI. Μετρήσεις MPI κώδικα.

Με allreduce :

			Pure Mpi						
ΧΡΟΝΟΙ									
Size of matrix/Διεργασίες	1	4	9	16	25	36	49	64	80
840x840	0,7951	0,2127	0,1084	0,0751	0,0572	0,0688	0,0689	0,0434	0,0620
1680x1680	3,1722	0,8378	0,3937	0,2267	0,1560	0,1407	0,1354	0,0878	0,1049
3360x3360	12,6812	3,2995	1,4980	0,9208	0,5594	0,4158	0,3297	0,2488	0,2209
6720x6720	50,7138	13,0593	5,8278	3,6043	2,1642	1,5186	1,1843	0,9527	0,7698
13440x13440	213,3130	52,0971	23,1955	33,5897	8,4642	5,9158	4,4362	3,6659	2,9495
26880x26880	853,3510	215,3130	92,4546	56,7218	33,8099	23,4369	17,4265	14,3092	11,3794

Speedup									
Size of matrix/Διεργασίες		4	9	16	25	36	49	64	80
840x840	-	3,737	7,338	10,594	13,903	11,560	11,540	18,318	12,831
1680x1680	-	3,786	8,057	13,993	20,338	22,541	23,430	9,539	30,249
3360x3360	-	3,843	8,465	13,773	22,669	30,499	38,460	50,976	57,401
6720x6720	-	3,883	8,702	14,071	23,433	33,395	42,822	53,233	65,882
13440x13440	-	4,095	9,196	6,351	25,202	36,058	48,085	58,188	72,321
26880x26880	-	3,963	9,230	15,045	25,240	36,411	48,968	59,637	74,991

Efficiency									
Size of matrix/Διεργασίες		4	9	16	25	36	49	64	80
840x840	-	0,93	0,82	0,66	0,56	0,32	0,24	0,29	0,16
1680x1680	-	0,95	0,90	0,87	0,81	0,63	0,48	0,15	0,38
3360x3360	-	0,96	0,94	0,86	0,91	0,85	0,78	0,80	0,72
6720x6720	-	0,97	0,97	0,88	0,94	0,93	0,87	0,83	0,82
13440x13440	-	1,02	1,02	0,40	1,01	1,00	0,98	0,91	0,90
26880x26880	-	0,99	1,03	0,94	1,01	1,01	1,00	0,93	0,94

Παρατηρούμε ότι οι χρονοι πέφτουν δραματικά οταν εφαρμόζεται MPI παραλληλοποίηση. Από το efficiency ευκολα φαίνεται ότι σε μικρά προβλήματα τα πάρα πολλά process δεν αποτελούν καλή στρατηγική καθώς το κόστος επικοινωνίας είναι μεγαλύτερο από το κέρδος των παράλληλων υπολογισμών. Ωστόσο όσο το πρόβλημα μεγαλώνει φαίνεται η αξία του παραλληλισμού καθώς εκεί το speedup και το efficiency έχουν πολύ καλύτερες τιμές οδηγώντας στην διαπίστωση ότι το πρόγραμμα κλιμακώνει. Ενδεικτικά για το μικρότερο πρόβλημα 840x840 παρατηρούμε στο profiling που περιλαμβάνεται στον φάκελο ParallelMPI οτι το κόστος επικοινωνίας για 4 διεργασίες είναι μικρό ποσοστό επί του συνολικού χρόνου, αλλά για 64 διεργασίες είναι υπερβολικά μεγάλο ποσοστό του συνολικού χρόνου και αυτό αποτυπώνεται καλα στο efficiency. Για μεγαλύτερα προβλήματα όπως το 13440 το ποσοστό της επικοινωνίας μειώνεται.
Πρέπει να σημειωθεί ότι λόγω δεσμεύσεων σε μνήμη το μεγαλύτερο πρόβλημα 26880x26880 δεν μπορέσαμε να το μετρήσουμε σε 4 process καθως δεν επιτρεπόταν η δέσμευση μνήμης για τους υποπίνακες.

Χωρις allreduce :

			Pure Mpi =NO ALLREDUCE						
ΧΡΟΝΟΙ									
Size of matrix/Διεργασίες	1	4	9	16	25	36	49	64	80
840x840	0,7951	0,2123	0,1136	0,0670	0,0662	0,0593	0,0634	0,0395	0,0610
1680x1680	3,1722	0,8355	0,3978	0,2327	0,1611	0,1391	0,1270	0,0947	0,9049
3360x3360	12,6812	3,2997	1,4670	0,9165	0,5677	0,4203	0,3494	0,2280	0,2109
6720x6720	50,7138	13,0661	5,8266	3,6063	2,1517	1,5201	1,1646	0,2541	0,7798
13440x13440	213,3130	52,1010	23,1421	14,2340	8,4695	5,9293	4,4258	3,6343	2,8495
26880x26880	853,3510	213,3130	92,3770	56,7771	33,4926	23,4220	17,4052	14,2971	11,2794

Speedup									
Size of matrix/Διεργασίες		4	9	16	25	36	49	64	80
840x840	-	3,745	6,999	11,870	12,014	13,415	12,537	20,115	13,035
1680x1680	-	3,797	7,974	13,635	19,688	22,804	24,978	8,825	3,506
3360x3360	-	3,843	8,644	13,836	22,338	30,171	36,293	55,619	60,122
6720x6720	-	3,881	8,704	14,062	23,569	33,363	43,545	199,575	65,037
13440x13440	-	4,094	9,218	14,986	25,186	35,976	48,197	58,695	74,859
26880x26880	-	4,000	9,238	15,030	25,479	36,434	49,028	59,687	75,656

Efficiency									
Size of matrix/Διεργασίες		4	9	16	25	36	49	64	80
840x840	-	0,94	0,78	0,74	0,48	0,37	0,26	0,31	0,16
1680x1680	-	0,95	0,89	0,85	0,79	0,63	0,51	0,14	0,04
3360x3360	-	0,96	0,96	0,86	0,89	0,84	0,74	0,87	0,75
6720x6720	-	0,97	0,97	0,88	0,94	0,93	0,89	3,12	0,81
13440x13440	-	1,02	1,02	0,94	1,01	1,00	0,98	0,92	0,94
26880x26880	-	1,00	1,03	0,94	1,02	1,01	1,00	0,93	0,95

Εδώ παρατηρούμε μικρή διαφορά (στα δεκαδικά κυρίως ψηφία) η οποία οφείλεται στο κόστος της MPI_Allrecude, καθώς η συγκεκριμένη εντολή προσθέτει κόστος επικοινωνίας. Ενδεικτικά, στο πρόβλημα 13440x13440 φαίνεται αρκετή αλλαγή στο efficiency που αυξάνεται από 0,90 σε 0,94.

CHALLENGE

			CHALLENGE						
XPONOI									
Size of matrix/Διεργασίες	1	4	9	16	25	36	49	64	80
840x840	0,8364	0,2257	0,1365	0,0965	0,0985	0,1085	0,1106	0,0944	0,1033
1680x1680	3,3300	0,8719	0,4408	0,2926	0,2593	0,2233	0,2314	0,2453	0,2914
3360x3360	13,3052	3,4255	1,6512	1,0468	0,8357	0,6885	0,5654	0,5110	0,5096
6720x6720	53,2076	13,6682	6,4735	4,0228	3,0495	2,2383	2,0085	2,0898	1,6150
13440x13440	213,3130	-	-	20,8579	11,9115	9,2774	7,6094	6,6432	5,9153
26880x26880	853,3510	-	-	-	-	-	-	-	-

Speedup									
Size of matrix/Διεργασίες		4	9	16	25	36	49	64	80
840x840	-	3,705	6,125	8,663	8,493	7,708	7,560	8,860	8,096
1680x1680	-	3,819	7,555	11,382	12,841	14,913	14,393	3,554	11,428
3360x3360	-	3,884	8,058	12,710	15,920	19,325	23,531	26,040	26,111
6720x6720	-	3,893	8,219	13,226	17,448	23,772	26,491	25,460	32,945
13440x13440	-	-	-	10,227	17,908	22,993	28,033	32,110	36,061
26880x26880	-	-	-	-	-	-	-	-	-

Efficiency									
Size of matrix/Διεργασίες		4	9	16	25	36	49	64	80
840x840	-	0,93	0,68	0,54	0,34	0,21	0,15	0,14	0,10
1680x1680	-	0,95	0,84	0,71	0,51	0,41	0,29	0,06	0,14
3360x3360	-	0,97	0,90	0,79	0,64	0,54	0,48	0,41	0,33
6720x6720	-	0,97	0,91	0,83	0,70	0,66	0,54	0,40	0,41
13440x13440	-	-	-	0,64	0,72	0,64	0,57	0,50	0,45
26880x26880	-	-	-	-	-	-	-	-	-

ΣΥΓΚΡΙΣΗ ΜΕ CHALLENGE

Διαφορά χρόνων με Challenge (OUR_TIME - CHALLENGE_TIME)									
Size of matrix/Διεργασίες	1	4	9	16	25	36	49	64	80
840x840	-0,0412	-0,0130	-0,0282	-0,0215	-0,0413	-0,0397	-0,0417	-0,0510	-0,0413
1680x1680	-0,1578	-0,0341	-0,0470	-0,0659	-0,1033	-0,0826	-0,0960	-0,1575	-0,1865
3360x3360	-0,6240	-0,1260	-0,1532	-0,1260	-0,2763	-0,2727	-0,2357	-0,2622	-0,2886
6720x6720	-2,4938	-0,6089	-0,6457	-0,4186	-0,8853	-0,7197	-0,8242	-1,1371	-0,8453
13440x13440	-	-	-	12,7318	-3,4473	-3,3616	-3,1733	-2,9773	-2,9658
26880x26880	-	-	-	-	-	-	-	-	-

Κοιτάζοντας τον τελευταίο πίνακα με την διαφορά των δύο μετρήσεων εύκολα παρατηρούμε ότι οι βελτιστοποιήσεις που έγιναν στο ακολουθιακό τμήμα αλλά και στο mpi (επικαλυψη της επικοινωνιας, μεθοδολογια foster κ.α.) έδωσαν σαν αποτέλεσμα ένα πιο δυνατό πρόγραμμα σε θέμα ταχύτητας από αυτό του challenge.

VI. Υβριδικό MPI+OpenMp.

Στο υβριδικό MPI+OpenMp σε κάθε διεργασία δημιουργούνται νήματα. Τα νήματα δημιουργούνται σε κάθε διεργασία πριν την κεντρική επανάληψη, όπως στις οδηγίες στο σημείο 16 έτσι ώστε να μην δημιουργούνται και καταστρέφονται συνεχώς και καθυστερεί ο χρόνος. Αρχικά δοκιμάσαμε ένα #omp parallel for για κάθε μία από τις 5 επαναλήψεις (ένα για το διπλό for και από ένα για κάθε εξωτερική σειρά/στήλη). Η λύση στην οποία καταλήξαμε ήταν να βάλουμε #omp parallel for έξω από την κεντρική επανάληψη ώστε να δημιουργούνται τα threads και έπειτα #omp master ώστε η κεντρική να τρέχει μόνο από το κύριο νήμα. Οι κλήσεις mpi γίνονται επίσης μόνο από το κύριο νήμα ενώ στις εσωτερικές επαναλήψεις τα νήματα διανείμονται με #omp for. Έχουμε χρησιμοποιήσει πολλά barriers επειδή χρειάζεται μετά από κάθε mpi_receive που κάνει το κύριο νήμα να περιμένουν όλα τα νήματα το αποτέλεσμα και, γενικά, απαιτείται πολύς συγχρονισμός των νημάτων.

VII. Μετρήσεις MPI+OPENMP κώδικα.

	Hybrid openmp-mpi					
ΧΡΟΝΟΙ						
Size of matrix/Διεργασίες-Νήματα	1	2-4	8-16	18-36	32-64	40-80
840x840	0,7951	0,2038	0,0593	0,0855	0,0596	0,1221
1680x1680	3,1722	0,8213	0,2273	0,1110	0,0897	0,1286
3360x3360	12,6812	6,4240	0,9313	0,4018	0,5120	0,2434
6720x6720	50,7138	12,8864	3,5793	1,5113	0,9528	0,7794
13440x13440	213,3130	51,3604	28,0028	5,8923	3,6403	2,9323
26880x26880	853,3510	-	56,3418	23,2662	14,2844	11,5706

Size of matrix/Διεργασίες-Νήματα	1	2-4	8-16	18-36	32-64	40-80
840x840	-	3,9009	13,4112	9,2998	13,3515	6,5106
1680x1680	-	3,8622	13,9573	28,5853	35,3523	24,6614
3360x3360	-	1,9740	13,6167	31,5632	24,7675	52,1007
6720x6720	-	3,9354	14,1686	33,5555	53,2250	65,0693
13440x13440	-	4,1533	7,6175	36,2018	58,5970	72,7468
26880x26880	-	-	15,1460	36,6777	59,7403	73,7517

Efficiency						
Size of matrix/Διεργασίες-Νήματα	1	2-4	8-16	18-36	32-64	40-80
840x840	-	0,9752	0,8382	0,2583	0,2086	0,0814
1680x1680	-	0,2414	0,3877	0,4466	0,4419	0,3083
3360x3360	-	0,4935	0,8510	0,8768	0,3870	0,6513
6720x6720	-	0,9839	0,8855	0,9321	0,8316	0,8134
13440x13440	-	1,0383	0,4761	1,0056	0,9156	0,9093
26880x26880	-	-	0,9466	1,0188	0,9334	0,9219

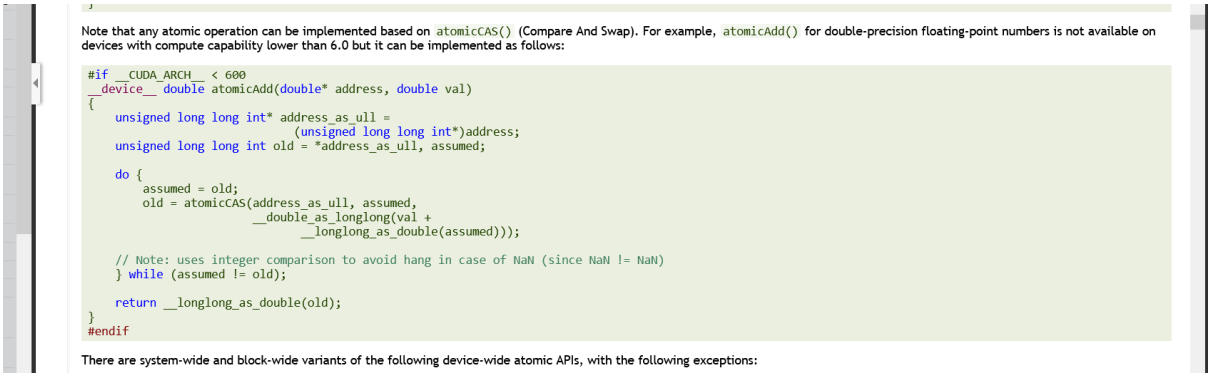
Οπως φαίνεται στον πίνακα με την χρήση omp threads καταφέραμε να ρίξουμε τον χρόνο συγκριτικά με το καθαρό MPI καθώς τώρα με 2 processes και 2 threads ανά

process έχουμε ελαφρώς πιο μικρό χρονο απο 4 processes με pure MPI. Ωστόσο δεν θα μπορούσαμε να οδηγηθούμε σε διαφορετικά αποτελέσματα όσον αφορά την κλιμάκωση και το κόστος επικοινωνίας καθώς τον χρόνο που κερδίζουμε απο την μισή επικοινωνία που γίνεται τώρα για το MPI, τον χάνουμε για τον συγχρονισμό των threads.

VIII. CUDA 1 με GPU.

Για την υλοποίηση σε 1 GPU, κύριο μέρος αποτελεί η δέσμευση μνήμης για τους δύο πίνακες u και u_old καθώς και για το loop_error κατευθείαν στην μνήμη της GPU με τις κατάλληλες εντολές όπως φαίνεται στον κώδικα. Στην συνέχεια για την δημιουργία του grid μπορούν να χρησιμοποιηθουν 4, 8, 16 η max 32 blocks ανά διάσταση λόγω των περιορισμών μνήμης και έτσι οι παρακάτω μετρήσεις είναι με 32 που ειναι προφανώς το πιο γρήγορο.

Για την συνάρτηση jacobi αρχικά υπολογίζουμε το index που αποτελεί και το id του thread στο block καθώς και τα x και y που αποτελούν τα index για το κάθε φορά row και column, τις συντεταγμένες δηλαδή του στοιχείου που θα υπολογίσει στη συνεχεια το κάθε thread στον πίνακα. Μετά τον υπολογισμό ακολουθεί το reduction για το error όπως παρουσιάζεται και στο guide βάζοντας το άθροισμα των error κάθε block στο temp[0]. Για το reduction μεταξύ block θα μπορούσε να χρησιμοποιηθεί κάποιος πίνακας που θα είχε δεσμεύσει μνήμη στην συσκευή αντί αυτού μέσα από το guide προέκυψε μια κομψότερη λύση και πολύ πιο γρήγορη μέσω της συνάρτησης atomicadd. Για να λειτουργήσει η atomicadd για τύπο double έπρεπε να γίνει overloaded για double values.



Note that any atomic operation can be implemented based on `atomicCAS()` (Compare And Swap). For example, `atomicAdd()` for double-precision floating-point numbers is not available on devices with compute capability lower than 6.0 but it can be implemented as follows:

```
#if __CUDA_ARCH__ < 600
__device__ double atomicAdd(double* address, double val)
{
    unsigned long long int* address_as_ull =
                              (unsigned long long int*)address;
    unsigned long long int old = *address_as_ull, assumed;

    do {
        assumed = old;
        old = atomicCAS(address_as_ull, assumed,
                        __double_as_longlong(val +
                               __longlong_as_double(assumed)));

    // Note: uses integer comparison to avoid hang in case of NaN (since NaN != NaN)
    } while (assumed != old);

    return __longlong_as_double(old);
}
#endif
```

There are system-wide and block-wide variants of the following device-wide atomic APIs, with the following exceptions:

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomic-functions
φυσικα αυτο στην περιπτωση μας δεν μπορουσε να δουλεψει αφου δημιουργόταν το ερρορ :
error: function "atomicAdd(double *, double)" has already been defined
αυτο ομως επιλυθηκε απο την εξης πηγη :
https://stackoverflow.com/questions/39274472/error-function-atomicadddouble-double-has-already-been-defined

VIII.Μετρήσεις CUDA με 1 GPU.

CUDA		
Size of matrix/Διεργασίες	Sequential	CUDA 1 GPU
840x840	0,7951	0,0223
1680x1680	3,1722	0,0774
3360x3360	12,6812	0,2892
6720x6720	50,7138	0,9644
13440x13440	213,3130	3,2546
26880x26880	853,3510	12,5661

Speedup		
Size of matrix/Διεργασίες		CUDA 1 GPU
840x840	-	35,697
1680x1680	-	40,992
3360x3360	-	43,843
6720x6720	-	52,583
13440x13440	-	65,541
26880x26880	-	67,909

Εδω παρατηρουμε εντυπωσιακές διαφορές, η χρήση και μιας μόνο GPU έχει πάρα πολύ μεγάλη επιτάχυνση για τον αλγόριθμο του jacobi η διαφορα των χρονων που φαινεται παραπανω ειναι εξαιρετικα μεγαλυτερη σε σχεση με το mpi και hybrid .Το ιδιο μπορουμε να πουμε και για την κλιμακωση καθως οσο ανεβαινει το προβλημα το ιδιο κανει και το speedup. Οι συγκεκριμενες μετρησεις εγιναν με 32*32 blocks και `((numerator+denominator-1)/denominator)*((numerator+denominator-1)/denominator)` thread. Ειναι αντιληπτο οτι το κοστος που υπαρχει για την επικοινωνια των gpu thread ειναι εξαιρετικα μικροτερο απο οποιαδηποτε αλλη επικοινωνια μελετήσαμε μεχρι εδω process η cpu thread .

VIIII. CUDA με 2 GPU.

Έγιναν μικρές αλλαγές από την υλοποίηση με 1 GPU. Τώρα χωρίζουμε το πρόβλημα στην μέση, ώστε κάθε GPU να αναλαμβάνει να λύσει το μισό. Έτσι αναμένουμε σχεδόν υποδιπλασιασμό του χρόνου, καθώς οι GPUs κάνουν τους μισούς υπολογισμούς η κάθε μία παράλληλα με την άλλη αλλά υπάρχει και ένα κόστος για την ανταλλαγή της άλω που στην συγκεκριμένη περίπτωση είναι μόνο 1 σειρά για κάθε υποπίνακα.
Ειδικότερα μερικές από τις αλλαγές είναι ότι τώρα δεσμεύουμε 4 υποπινακες, 2 για κάθε GPU. Μέσα στην κεντρική επανάληψη γίνεται πρώτα η επικοινωνία παρόμοια με το MPI δηλαδή γίνονται με memcpy ,send και receive τα απαραίτητα στοιχεία, η κάτω δηλαδή σειρά του πίνακα στην πρώτη GPU γίνεται send στην δεύτερη και αντίστοιχα η πάνω της δεύτερης GPU στην κάτω της πρώτης (τοπολογία πάνω 0 - κάτω 1). Στην συνέχεια δημιουργούνται 2 openmp threads, ένα για τον υπολογισμό των υποπινακων της πρώτης GPU και ένα για της δεύτερης, ώστε οι πυρήνες να κληθούν παράλληλα. Μετά τον υπολογισμό της jacobi από τα GPUs αθροίζουμε το συνολικό error με reduction στα threads και εν συνεχεία κάνουμε την διαίρεση για τον υπολογισμό του residual. Αξιοσημείωτο είναι πως στην πρώτη προσπάθεια βάλαμε τα thread να ξεκινάνε έξω από την κεντρική επανάληψη για να αποφύγουμε το overhead που προκαλεί η συνεχής δημιουργία και καταστροφή των thread σε κάθε επανάληψη, ωστόσο τα αποτελέσματα ήταν πολύ χειρότερα καθώς το κόστος των barrier για τον συγχρονισμό και σωστό υπολογισμό φαίνεται να είναι μεγαλύτερο. Το τελευταίο θα μπορούσε να αποτελεί καλή πρακτική όταν έχουμε παραπάνω GPUS και συνεπώς παραπάνω thread άλλα για τα δεδομένα της εργασίας η ταχύτερη επιλογή είναι να δημιουργούνται και να καταστρέφονται κάθε φορά για τον υπολογισμό της συνάρτησης jacobi.

Χ .Μετρήσεις CUDA με 2 GPU.

CUDA 2 GPU		
Size of matrix/Διεργασίες	Sequential	CUDA 2 GPU
840x840	0,7951	0,0130
1680x1680	3,1722	0,0421
3360x3360	12,6812	0,1592
6720x6720	50,7138	0,5596
13440x13440	213,3130	1,8318
26880x26880	853,3510	6,8955
Speedup		
Size of matrix/Διεργασίες		CUDA 2 GPU
840x840	-	61,228
1680x1680	-	75,360
3360x3360	-	79,665
6720x6720	-	90,621
13440x13440	-	116,453
26880x26880	-	123,755

Όπως ήταν αναμενόμενο ο χρόνος με 2 GPU είναι σχεδόν μισός από αυτόν με 1 GPU. Υπάρχει ένα μικρό κόστος επικοινωνίας το οποίο δεν είναι σημαντικό σε σχέση με την βελτίωση του χρόνου. Ο σχολιασμός περι scalability παραμένει ίδιος με του προγραμματος για 1 GPU.

XI. Συμπεράσματα.

Σε μικρά μεγέθη, βλέπουμε πολύ μικρότερη αποδοτικότητα για μεγάλο αριθμό process λόγω του overhead της επικοινωνίας τους, όπως και αναμένεται. Όσο αυξάνουμε τα μεγέθη, η αποδοτικότητα (efficiency) γίνεται όλο και καλύτερη μιάς και τα περισσότερα process οδηγούν σε πιο γρήγορη εκτέλεση,και το overhead της επικοινωνίας γίνεται αμελητέο σε σχέση με τη βελτίωση στο χρόνο εκτέλεσης.
Από ένα σημείο και μετά η αύξηση διεργασιών δίνει πολύ μικρή αποδοτικότητα λόγω του νόμου του amdahl, δηλαδή λόγω του μη παραλληλοποιήσιμου κομματιού του προγράμματος.
Μικρότερη μείωση στο efficiency παρατηρούμε καθώς αυξάνουμε νήματα και διεργασίες στη συμπεριφορά του hybrid από αυτή του mpi.
Δεν υπάρχει σημαντική διαφορά ανάμεσα στις κλήσεις με allreduce και reduce και σε αυτές χωρίς επι το πλείστον, αν και εμφανίζονται κάποιες μειώσεις.Οι διακυμάνσεις που παρατηρήθηκαν ήταν αμελητέες και μπορεί να οφείλονται στο τότε φόρτο του argo συστήματος.

*****τα κυανα κομματια των πινακων ειναι προσεγγιστικες τιμες καθως δεν μπορουσαν να δημιουργηθουν οι απαραιτητοι πινακες του προγραμματος λογω ελλειψης μνημης*

Όσον αφορά την παραλληλοποίηση σε GPU και μονό από τις μετρήσεις φαίνεται η εξαιρετική διάφορα στους χρόνους τόσο με 1 GPU όσο και με 2 όπου στην τελευταία περίπτωση οι χρόνοι είναι περίπου στο μισό όπως και αναμενότα. Μικρή εξαίρεση αποτελούν τα πολύ μικρά προβλήματα όπου το overhead της επικοινωνίας των 2 GPU έχουν αντίκτυπο στον χρόνο όχι όμως κάτι απαγορευτικό.

Ενδεικτικά μερικά ακόμα profilings (υπάρχουν και κάποια στον φάκελο του mpi τα οποία έχουν σχολιαστεί πιο πάνω):

4 procs 840x840

```
@ mpiP
@ Command : mpijac.x
@ Version                  : 3.5.0
@ MPIP Build date          : Nov 11 2020, 13:45:15
@ Start time               : 2021 10 11 18:56:56
@ Stop time                : 2021 10 11 18:56:56
@ Timer Used               : PMPI_Wtime
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 1475131
@ Final Output Dir         : .
@ Report generation        : Single collector task
@ MPI Task Assignment      : 0 argo-c0
@ MPI Task Assignment      : 1 argo-c0
@ MPI Task Assignment      : 2 argo-c0
@ MPI Task Assignment      : 3 argo-c0

---------------------------------------------------------------------------
@--- MPI Time (seconds) ------------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0      0.218      0.029    13.30
   1      0.215     0.0293    13.59
   2      0.218     0.0302    13.81
   3      0.216     0.0314    14.56
   *      0.868       0.12    13.81
---------------------------------------------------------------------------
@--- Callsites: 124 ----------------------------------------------------------
```

```
124   0 0x14f1f2986c7a          [unknown]                Isend
---------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
---------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%      Count    COV
Cart_create           113       9.97    1.15    8.32          1   0.00
Allreduce             102       9.94    1.15    8.29         50   0.00
Cart_create            20       9.93    1.14    8.29          1   0.00
Allreduce               9       9.89    1.14    8.25         50   0.00
Bcast                  46       8.77    1.01    7.31          1   0.00
Allreduce              71       7.62    0.88    6.35         50   0.00
Allreduce              40       7.53    0.87    6.28         50   0.00
Bcast                  78       6.77    0.78    5.65          1   0.00
Cart_create            51       3.62    0.42    3.02          1   0.00
Cart_create            82       3.61    0.42    3.02          1   0.00
Barrier                73       2.99    0.34    2.49          1   0.00
Barrier                42       2.96    0.34    2.47          1   0.00
Wait                   61        2.8    0.32    2.34         50   0.00
Wait                  114       2.71    0.31    2.26         50   0.00
Wait                   21        2.7    0.31    2.25         50   0.00
Wait                   92       1.71    0.20    1.43         50   0.00
Irecv                  77       1.51    0.17    1.26         50   0.00
Bcast                 108       1.37    0.16    1.14          1   0.00
Isend                  18       1.02    0.12    0.85         50   0.00
Isend                 111      0.988    0.11    0.82         50   0.00
---------------------------------------------------------------------------
```

16 procs 3360x3360

```
@ mpiP
@ Command : mpijac.x
@ Version                  : 3.5.0
@ MPIP Build date          : Nov 11 2020, 13:45:15
@ Start time               : 2021 10 11 19:01:00
@ Stop time                : 2021 10 11 19:01:01
@ Timer Used               : PMPI_Wtime
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 1476168
@ Final Output Dir         : .
@ Report generation        : Single collector task
@ MPI Task Assignment      : 0 argo-c0
@ MPI Task Assignment      : 1 argo-c0
@ MPI Task Assignment      : 2 argo-c0
@ MPI Task Assignment      : 3 argo-c0
@ MPI Task Assignment      : 4 argo-c0
@ MPI Task Assignment      : 5 argo-c0
@ MPI Task Assignment      : 6 argo-c0
@ MPI Task Assignment      : 7 argo-c0
@ MPI Task Assignment      : 8 argo-c1
@ MPI Task Assignment      : 9 argo-c1
@ MPI Task Assignment      : 10 argo-c1
@ MPI Task Assignment      : 11 argo-c1
@ MPI Task Assignment      : 12 argo-c1
@ MPI Task Assignment      : 13 argo-c1
@ MPI Task Assignment      : 14 argo-c1
@ MPI Task Assignment      : 15 argo-c1

---------------------------------------------------------------------------
```

```
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
--------------------------------------------------------------------------
Call                Site       Time    App%    MPI%      Count    COV
Allreduce            287       50.6    0.33    3.13         50   0.00
Allreduce             71       48.2    0.32    2.98         50   0.00
Allreduce            134       47.9    0.32    2.96         50   0.00
Allreduce              9       47.4    0.31    2.93         50   0.00
Allreduce            380       44.9    0.30    2.78         50   0.00
Allreduce            194       44.7    0.30    2.76         50   0.00
Allreduce            227       41.8    0.28    2.59         50   0.00
Allreduce             40       41.2    0.27    2.55         50   0.00
Allreduce            320       39.7    0.26    2.46         50   0.00
Allreduce            256       39.4    0.26    2.44         50   0.00
Allreduce            164         38    0.25    2.35         50   0.00
Allreduce            441       37.7    0.25    2.33         50   0.00
Allreduce            410       34.4    0.23    2.13         50   0.00
Allreduce            351         34    0.22    2.10         50   0.00
Allreduce            472       33.2    0.22    2.05         50   0.00
Allreduce            100       30.7    0.20    1.90         50   0.00
Bcast                387       27.4    0.18    1.69          1   0.00
Bcast                264       22.7    0.15    1.40          1   0.00
Wait                 245       21.5    0.14    1.33         50   0.00
Bcast                295       21.4    0.14    1.32          1   0.00
--------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
--------------------------------------------------------------------------
Call                Site      Count      Total       Avrg  Sent%
Isend                329         50   3.36e+05   6.72e+03   1.56
Isend                 74         50   3.36e+05   6.72e+03   1.56
Isend                334         50   3.36e+05   6.72e+03   1.56
Isend                321         50   3.36e+05   6.72e+03   1.56
Isend                 88         50   3.36e+05   6.72e+03   1.56
Isend                 93         50   3.36e+05   6.72e+03   1.56
Isend                 80         50   3.36e+05   6.72e+03   1.56
Isend                341         50   3.36e+05   6.72e+03   1.56
Isend                105         50   3.36e+05   6.72e+03   1.56
Isend                360         50   3.36e+05   6.72e+03   1.56
Isend                365         50   3.36e+05   6.72e+03   1.56
Isend                111         50   3.36e+05   6.72e+03   1.56
Isend                352         50   3.36e+05   6.72e+03   1.56
Isend                124         50   3.36e+05   6.72e+03   1.56
Isend                383         50   3.36e+05   6.72e+03   1.56
Isend                372         50   3.36e+05   6.72e+03   1.56
Isend                119         50   3.36e+05   6.72e+03   1.56
```