ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ
ΕΡΓΑΣΙΑ 2019-2020

Ρέππας Γιάννης – 1115201500137

Τασσόπουλος Γιάννης - 1115201500155

GAME OF LIFE

Περιεχόμενα

Εισαγωγή

Στην εργασία υλοποιήθηκε ο αλγόριθμος Game of Life του John Conway. Το Game of Life περιλαμβάνει ένα γράφημα από τιμές 0 ή 1 σε έναν N x M πίνακα. Ανάλογα με τις αρχικές τιμές του πίνακα, κάθε τιμή/κελί αλλάζει τιμή ανάλογα με τις γειτονικές του τιμές/κελιά. Κάθε αλλαγή όλων των κελιών του πίνακα αναπαριστά και μία νέα «γενιά» όλων των κελιών. Στόχος της εργασίας ήταν να υλοποιηθεί ο παραπάνω αλγόριθμος με διάφορους τρόπους (π.χ. σειριακή και παράλληλη εκτέλεση) και να παρθούν συμπεράσματα όσον αφορά την καλύτερη χρονικά εκτέλεση με βάση τους πόρους που καταναλώνει και διαχειρίζεται ο υπολογιστής. Το Game of Life υλοποιήθηκε σε γλώσσα C και οι μετρήσεις έγιναν σε περιβάλλον linux, από τα μηχανήματα argo της σχολής. Πέραν της σειριακής εκτέλεσης η οποία έχει μόνο μία εκδοχή, το παράλληλο πρόγραμμα δίνει τη δυνατότητα στο χρήστη να προσθέσει και συνθήκη ελέγχουν allReduce, η οποία εξετάζει την περίπτωση που ο πίνακας παραμένει ίδιος σε δύο διαδοχικές γενιές. Επίσης, δίνεται η δυνατότητα ο χρήστης να κάνει εκτέλεση όχι μόνο με πολλαπλές διεργασίες (mpi), αλλά και με πολλά νήματα(omp). Όλη η εργασία υλοποιήθηκε με χαρά και διάθεση και από τα 2 μέλη.

Σχεδιασμός Διαμοιρασμού Δεδομένων

Αρχικά, όταν ο χρήστης εκτελεί το παράλληλο πρόγραμμα, γίνεται διαμοιρασμός του αρχικού πίνακα στις διεργασίες. Για N αριθμό διεργασιών, ο πίνακας χωρίζεται σε N υπο-πίνακες. Ο κάθε υπο-πίνακας περιέχει τα αντίστοιχα δεδομένα που του ανατέθηκαν από τον αρχικό πίνακα, καθώς και μία επιπλέον περίμετρο, δηλαδή αυξημένο μήκος κατά 1 και στις δύο διαστάσεις του. Κάτι τέτοιο συμβαίνει επειδή για τον υπολογισμό των περιμετρικών του στοιχείων για τη νέα γενιά απαιτούνται δεδομένα από γειτονικές διεργασίες, τα οποία παίρνονται μέσω επικοινωνίας μεταξύ των διεργασιών και αποθηκεύονται στην επιπλέον περίμετρο έτσι ώστε να γίνει στη συνέχεια ο υπολογισμός.

Ανάλογα με τον αριθμό των διεργασιών για τον οποίο γίνεται η εκτέλεση του προγράμματος, δημιουργείται και ο αντίστοιχος «πίνακας διεργασιών», με βάση τον μέγιστο κοινό διαιρέτη του αριθμού των διεργασιών και των διαστάσεων του αρχικού πίνακα. Αυτη η διαδικασία γίνεται με τη βοήθεια των συναρτήσεων MPI_Cart_create και MPI_Cart_coords. Η ανίχνευση των γειτονικών διεργασιών με βάση το id της παρούσας διεργασίας γίνεται με τη βοήθεια της συνάρτησης MPI_Cart_shift, αλλά και εκμεταλεύοντας και την ήδη γνωστή δομή του πίνακα διεργασιών. Πιο συγκεκριμένα:

0	1
2	3
4	5
6	7

Παράδειγμα για N = 8

Στο παραπάνω παράδειγμα, θα έχουμε για processRank = 7: up-> 5 , down-> 1, left-> 6, right-> 6, upleft-> 4, downleft-> 0, upright-> 4, downright-> 0

Σχεδιασμός MPI κώδικα

Σχδιασμός διαμοιρασμού δεδομένων και επικοινωνίας

Όσον αφορά αυτό το κομμάτι, ακολουθήθηκε η διαδικασία που αναφέρθηκε και στην εισαγωγή. Υλοποιήθηκαν datatypes για τη μεταφορά ολόκληρων σειρών και στηλών από τις γειτονικές διεργασίες.

Μείωση αδρανούς χρόνου και overhead

Αρχικά, υλοποιήθηκε ο αλγόριθμος που περιγράφεται και στις διαφάνειες. Πρώτα εκτελούνται τα requests και μετά τα send, τα οποία αρχικοποιούνται με MPI_Send_init και MPI_Recv_init εκτός της κεντρικής επανάληψης. Δεν χρησιμοποιούνται buffers για αντιγραφή και χρησιμοποιούνται datatypes για στήλες και γραμμές. Τέλος, οι γειτονικές διεργασίες βρίσκονται εκτός της κεντρικής επανάληψης και μία μόνο φορά. Προσέχουμε να γίνονται μέσα στο κεντρικό for loop όσο το δυνατόν λιγότερες πράξεις.

Γενικότερα, υλοποιήθηκαν όλα τα ζητούμενα 3-8 από τις οδηγίες που δοθήκαν για καλύτερο MPI κώδικα.

Μείωση ακολουθιακών υπολογισμών στην κεντρική επανάληψη

Χρησιμοποιήθηκαν δυναμικοί πίνακες, οι συναρτήσεις δηλώθηκαν ως inline, στο compilation χρησιμοποιήθηκε -O3.

Κάτι το οποίο δυστυχώς δεν υλοποιήθηκε, ήταν ο έλεγχος του reduce χωρίς διπλό for loop.

Σχεδιασμός υβριδικού κώδικα

Υλοποιήθηκαν οι 2 παραλληλοποιήσεις (12α,12β) που αναφέρονται στη διαφάνεια.

Μετρήσεις

Μετρήσεις για σειριακό πρόγραμμα

Το σειριακό πρόγραμμα περιλαμβάνει αντιγραφή όλων των στοιχείων από το u1 στο u2, κάτι το οποίο δε συμβαίνει στα άλλα 2 προγράμματα. Παρ' όλα αυτά, όπως φαίνεται και εξηγείται στη συνέχεια, η διαφορά με το πρόγραμμα mpi και υβριδικό mpi omp είναι εμφανής.

Grid Size	Time (seconds)
320 x 320	0.657312
640 x 640	2.608709
1280 x 1280	10.379222
2560 x 2560	41.184350
5120 x 5120	164.408886

Μετρήσεις για πρόγραμμα MPI με allReduce

Για το πρόγραμμα MPI είχαμε σε όλες τις εκτελέσεις ενεργοποιημένο το allReduce ανά 10 επαναλήψεις/γενιές. Πολλές μετρήσεις έγιναν για ίδιο αριθμό εργασιών αλλά για διαφορετικό αριθμό κόμβων. Ενδεχομένως θα μπορούσαμε να είχαμε επεκταθεί και άλλο στο τελευταίο κομμάτι για κάποιους αριθμούς διεργασιών, ωστόσο δυστυχώς λόγω έλλειψης χρόνου και όγκου αποτελεσμάτων αποφύγαμε κάτι τέτοιο. Όπως και να έχει, πάρθηκαν συμπεράσματα για τις εκτελέσεις.

Ως Speedup ορίζεται το πηλίκο του χρόνου εκτέλεσης για μία διεργασία με το τον χρόνο εκτέλεσης για τη παρούσα εκτέλεση που εξετάζουμε.

Ως Efficiency ορίζεται το πηλίκο του speedup με τον συνολικό αριθμό διεργασιών.

Παράδειγμα εκτέλεσης:

mpirun ./game_of_life_mpi.exe -n 320 -m 320 -g 300 -re 10 -r 1 -peg 0

, όπου: -n ο αριθμός των γραμμών του πίνακα

-m ο αριθμός των στηλών του πίνακα

-g ο αριθμός των γενεών που θα υπολογιστούν (300 σε όλες τις μετρήσεις)

-re ο αριθμός των επαναλήψεων/γενεών ανά τις οποίες θα εκτελεστεί ο έλεγχος του allReduce (για -r 0 προφανώς δε λαμβάνεται υπόψιν)

-r το flag για την ενεργοποίηση του allReduce (πάντα ενεργοποιημένο, από τη στιγμή που βρίσκουμε ίδιο πίνακα, ο έλεγχος σταματάει και οι επαναλήψεις συνεχίζουν)

-peg το flag για την ενεργοποίηση της εκτύπωσης των αποτελεσμάτων ανά γενιά στο αρχείο results.txt (δε χρησιμοποιήθηκε για τις μετρήσεις, η εκτύπωση σε οθόνη/αρχείο είναι αρκετά χρονοβόρα)

Για τα διαγράμματα μετά από τους πίνακες μετρήσεων, επιλέγουμε την καλύτερη μέτρηση για ίδιο αριθμό διεργασιών.

Ακολουθούν εκτελέσεις για τα 5 διαφορετικά μεγέθη πίνακα:

320 x 320

Processes	Nodes	Time (seconds)	Speedup	Efficiency
1	1	0.353968	1	1
2	1	0.181191	1.953	0.9765
2	2	0.202552	1.747	0.8735
4	1	0.090715	3.901	0.97525
4	2	0.128194	2.761	0.69025
4	4	0.128965	2.744	0.686
8	1	0.047762	7.411	0.926375
8	2	0.072239	4.899	0.612375
8	4	0.090050	3.9307	0.4913375
8	8	4.959100	0.07137	0.00892125
16	2	0.069173	5.1171	0.31981875
16	4	0.081322	4.3526	0.2720375
16	8	0.080663	4.3882	0.2742625
32	4	0.065522	5.4022	0.16881875
32	8	0.075748	4.6729	0.146028125
64	8	0.102750	3.4449	0.0538265625

640 x 640

Processes	Nodes	Time (seconds)	Speedup	Efficiency
1	1	1.378715	1	1
2	1	0.695860	1.9813	0.99065
2	2	0.720615	1.9132	0.9566
4	1	0.358685	3.8438	0.96095
4	2	0.398476	3.4599	0.864975
4	4	0.374616	3.6803	0.920075
8	1	0.185459	7.4340	0.92925
8	2	0.208293	6.6191	0.8273875
8	4	0.223940	6.1566	0.769575
8	8	0.226352	6.0910	0.761375
16	2	0.123671	11.1482	0.6967625
16	4	0.135456	10.1783	0.63614375
16	8	0.139161	9.90733	0.619208125
32	4	0.087950	15.6761	0.489878125
32	8	0.097837	14.0919	0.440371875
64	8	0.111752	12.3372	0.19276875

1280 x 1280

Processes	Nodes	Time (seconds)	Speedup	Efficiency
1	1	5.563059	1	1
2	1	2.827035	1.96780	0.9839
2	2	2.835642	1.96183	0.980915
4	1	1.409192	3.94769	0.9869225
4	2	1.445167	3.84942	0.962355
4	4	1.442066	3.85770	0.964425
8	1	0.706214	7.87729	0.98466125
8	2	0.731986	7.59995	0.94999375
8	4	0.751938	7.39829	0.92478625
8	8	0.749704	7.42034	0.9275425
16	2	0.392950	14.15716	0.8848225
16	4	0.403553	13.78520	0.861575
16	8	0.409598	13.58175	0.848859375
32	4	0.220001	25.28651	0.7902034375
32	8	0.230620	24.12218	0.753818125
64	8	0.156675	35.50699	0.55479671875

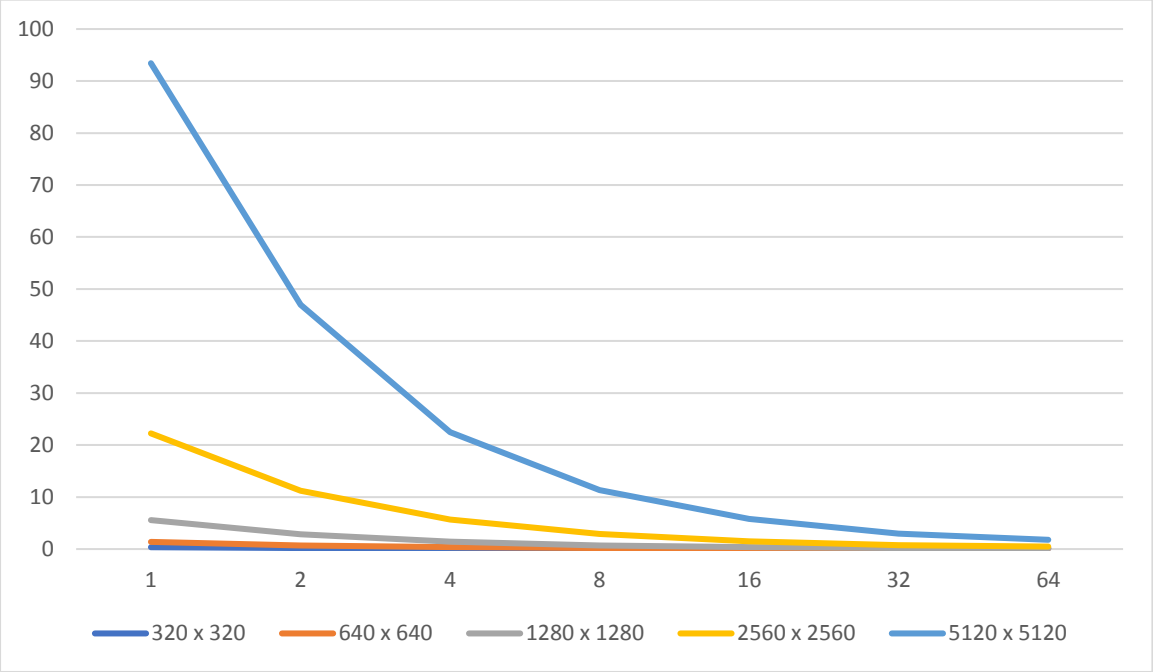
2560 x 2560

Processes	Nodes	Time (seconds)	Speedup	Efficiency
1	1	22.239957	1	1
2	1	11.219160	1.98231	0.991155
2	2	11.242592	1.97818	0.98909
4	1	5.685862	3.91144	0.97786
4	2	5.779194	3.84828	0.96207
4	4	5.757234	3.86295	0.9657375
8	1	3.230663	6.88402	0.8605025
8	2	2.904890	7.65604	0.957005
8	4	2.910966	7.64006	0.9550075
8	8	2.906224	7.65252	0.956565
16	2	1.643856	13.5291	0.84556875
16	4	1.492138	14.9047	0.93154375
16	8	1.467906	15.15080	0.946925
32	4	0.7576658	29.35325	0.9172890625
32	8	0.769292	28.90964	0.90342625
64	8	0.480712	46.26461	0.72288453125

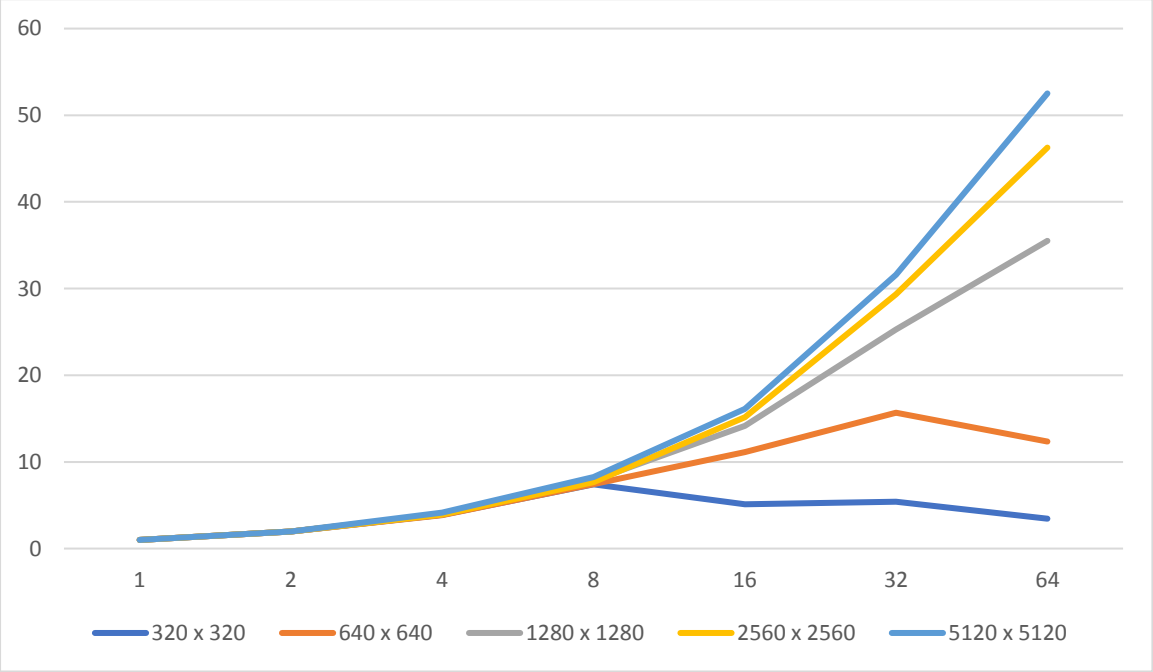
5120 x 5120

Processes	Nodes	Time (seconds)	Speedup	Efficiency
1	1	93.429083	1	1
2	1	47.202101	1.97934	0.98967
2	2	46.994955	1.98806	0.99403
4	1	22.454762	4.16076	1.04019
4	2	22.544361	4.14423	1.0360575
4	4	22.467035	4.15849	1.0396225
8	1	12.899711	7.24272	0.90534
8	2	11.754630	7.94827	0.99353375
8	4	11.356455	8.22695	1.02836875
8	8	11.324183	8.25040	1.0313
16	2	10.054505	9.29226	0.58076625
16	4	5.913451	15.79941	0.987463125
16	8	5.795326	16.12145	1.007590625
32	4	4.883090	19.13318	0.597911875
32	8	2.958677	31.57799	0.9868121875
64	8	1.779014	52.51733	0.82058328125

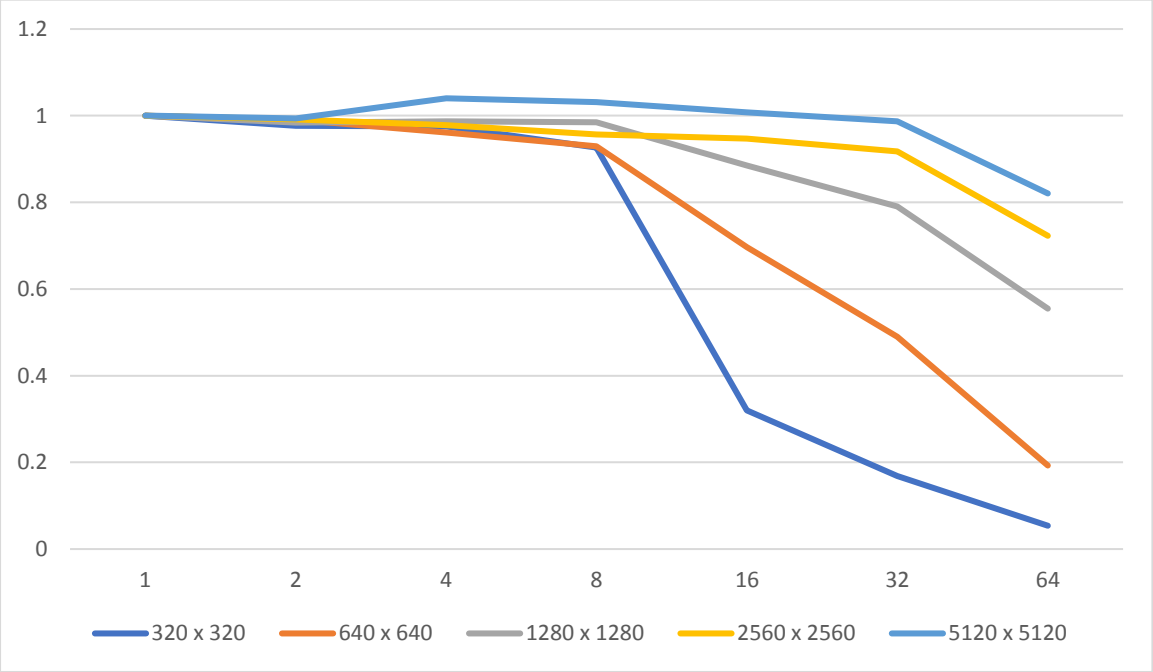
Διάγραμμα χρόνων εκτέλεσης



Διάγραμμα Speedup



Διάγραμμα Efficiency



Μετρήσεις για το υβριδικό πρόγραμμα MPI και Openmp

Για το υβριδικό πρόγραμμα, αρχικά δοκιμάζουμε για έναν κόμβο πιθανούς συνδυασμούς αριθμού διεργασιών και αριθμού νημάτων.

Επιλέξαμε συνδυασμούς 1δ-8ν , 1δ-16ν , 2δ-4ν , 2δ-8ν , 4δ-2ν , 4δ-4ν

Για μέγεθος 320 x 320 , έχουμε:

1δ-8ν	1δ-16ν	2δ-4ν	2δ-8ν	4δ-2ν	4δ-4ν
0.6708	0.78944	0.330035	0.3809	0.099295	> 5

Για μέγεθος 5120 x 5120, έχουμε:

2δ-4ν	2δ-8ν	4δ-2ν
>72	>72	18.34

Καταλήγουμε ότι 4δ-2ν είναι ο καταλληλότερος συνδυασμός

Έτσι λοιπόν για 1 κόμβο έχουμε ότι:

320 x 320	640 x 640	1280 x 1280	2560 x 2560	5120 x 5120
0.0929	0.3299	1.1614	4.7571	18.34

Για 4 κόμβους:

320 x 320	640 x 640	1280 x 1280	2560 x 2560	5120 x 5120
0.13	0.35	1.130	5.1	16.3777

Συμπεράσματα

Αρχικά, όσον αφορά το σειριακό πρόγραμμα, μπορούμε να εξάγουμε εύκολα συμπεράσματα κοιτώντας απλά τους 6 χρόνους εκτέλεσης (ένας για κάθε είδος πίνακα) και συγκρίνοντάς τους με αυτούς της μίας διεργασίας σε έναν κόμβο για το πρόγραμμα game_of_life_mpi.exe. Παρ' όλο που το πρόγραμμα του mpi χρησιμοποιεί μηνύματα μεταξύ διεργασιών (επί της ουσίας στέλνει και λαμβάνει μηνύματα από τον εαυτό του) ή εκτέλεση ελέγχου για ίδιο πίνακα σε διαδοχικές επαναλήψεις, με τους μετασχηματισμούς/κανόνες που εφαρμόσαμε (π.χ. χρήση δεικτών και όχι αντιγραφή του πίνακα σε κάθε loop) από το pdf με τις οδηγίες που μας δόθηκε, μπορούμε να δούμε ότι εξοικονομούμε χρόνο στον υπολογισμό των γενεών.

Όσον αφορά τις μετρήσεις που έγιναν για το πρόγραμμα game_of_life.exe, πραγματοποιήθηκαν δοκιμές για διαφορετικό μέγεθος πίνακα, αριθμό διεργασιών και αριθμό κόμβων (8 πυρήνες ανά κόμβο). Μία από τις πρώτες προφανείς παρατηρήσεις, είναι φυσικά η αύξηση του χρόνου εκτέλεσης για αυξανόμενο μέγεθος διαστάσεων του πίνακα. Όσο μεγαλύτερο είναι το μέγεθος του πίνακα, τόσο περισσότερες θα είναι και οι πράξεις που θα πρέπει να πραγματοποιηθούν και από το μηχάνημα. Στη συνέχεια, αλλά και κατά τη διάρκεια των μετρήσεων, μελετήσαμε τη «συμπεριφορά» του χρόνου εκτέλεσης όσον αφορά το διαφορετικό αριθμό των διεργασιών. Αυτό που παρατηρείται είναι ότι για μεγαλύτερο αριθμό από διεργασίες, έχουμε και πιο γρήγορες εκτελέσεις. Κάτι τέτοιο είναι λογικό, καθ' ότι το μηχάνημα αρχικά μπορεί και οργανώνει καλύτερα τις εκτελέσεις των πράξεων, παρ' όλο που απαιτείται επικοινωνία μεταξύ των διεργασιών, αλλά και φυσικά επειδή με τις παράλληλες εκτελέσεις των πράξεων γίνεται να μειωθεί η διάρκεια εκτέλεσης κατά πολύ. Αυτό φαίνεται άλλωστε και από τους χρόνους εκτέλεσης , αλλά και από τα διαγράμματα των χρόνων εκτέλεσης. Κάτι το οποίο ωστόσο αξίζει να σημειωθεί, είναι το γεγονός ότι για λίγες μόνο εκτελέσεις, ο χρόνος αυξάνεται, παρ' όλο που έχουμε αυξημένο αριθμό διεργασιών. Για παράδειγμα, βλέπουμε ότι στην περίπτωση των 8 κόμβων και των 64 διεργασιών για μικρό μέγεθος πίνακα (320 x 320), ο χρόνος εκτέλεσης αυξάνεται. Κάτι τέτοιο θεωρούμε ότι συμβαίνει για 2 λόγους: α) μικρό μέγεθος πίνακα και β) μεγάλος αριθμός διεργασιών. Σε περιπτώσεις όπου ο πίνακας χωρίζεται σε πολλές διεργασίες σε σχέση με το μικρό μέγεθός του,

είναι φυσικό ο υπολογιστής να σπαταλάει πολύ χρόνο για την επικοινωνία μεταξύ των διεργασιών και η κάθε διεργασία να μην έχει ιδιαίτερα μεγάλο όγκο από δεδομένα για να υπολογίσει. Κάτι τέτοιο μας οδηγεί στο εξής συμπέρασμα: Είναι απαραίτητο να βρίσκεται σε κάθε εκτέλεση, ανάλογα με το μέγεθος του πίνακα εισόδου, η «χρυσή τομή» του συνδυασμού διεργασίες/κόμβοι και μέγεθος πίνακα. Τέλος, όσον αφορά τους χρόνους εκτέλεσης του προγράμματος game_of_life_mpi.exe, υπάρχει μία λεπτομέρεια για την οποία δεν έχουμε σαφή εξήγηση. Αυτό συμβαίνει στην περίπτωση του πίνακα 320 x 320 για 8 κόμβους και 8 διεργασίες. Σε αυτήν την περίπτωση, ο χρόνος εκτέλεσης του προγράμματος εκτοξεύεται στα 5 δευτερόλεπτα. Αυτό που μπορούμε να υποθέσουμε είναι ότι επειδή όλες οι διεργασίες ανήκουν σε διαφορετικό κόμβο, η επικοινωνία μεταξύ τους είναι ιδιαίτερα κοστοβόρα και επιπλέον όπως αναφέρθηκε και πιο πριν, ο αριθμός των διεργασιών είναι αρκετά μεγάλος για το μέγεθος του πίνακα. Από την άλλη πλευρά, ο χρόνος εκτέλεσης δεν εκτοξεύεται τόσο για αντίστοιχες εκτελέσεις, επομένως ίσως υπήρχε κάποιο πρόβλημα στην εκτέλεση τη στιγμή που τρέχαμε το πρόγραμμα ,είτε κατά λάθος αλλάξαμε μόνοι μας μία παράμετρο.

Τις παραπάνω παρατηρήσεις μας όσον αφορά τη μείωση του χρόνου εκτέλεσης για μεγαλύτερο αριθμό διεργασιών (αλλά και την αύξηση σε μερικές περιπτώσεις που αναφέρθηκαν πιο πριν) έρχονται να επιβεβαιώσουν και τα διαγράμματα μετά τον υπολογισμό του speedup και efficiency. Χαρακτηριστικά, βλέπουμε στο διάγραμμα του speedup ότι οι εκτελέσεις για μεγαλύτερο μέγεθος πλέγματος έχουν μεγαλύτερο ρυθμό αύξησης του speedup σε αντίθεση με αυτές των μικρότερων πλεγμάτων. Επίσης, άλλη μια αναμενόμενη εικόνα στα διαγράμματα είναι και αυτή της αποδοτικότητας, η οποία σταδιακά μειώνεται, για όλα τα είδη πλέγματος, καθώς αυξάνεται ο αριθμός των διεργασιών.

Τέλος, λίγα λόγια και για τις εκτελέσεις του προγράμματος game_of_life_hybrid.exe. Όπως αναφέρθηκε και στην εκφώνηση, αρχικά επιλέγουμε να κάνουμε μετρήσεις για ένα μόνο κόμβο, δοκιμάζοντας διαφορετικούς συνδυασμούς διεργασιών και νημάτων. Τα μέρη του κώδικα τα οποία υλοποιήθηκαν παράλληλα με τη βιβλιοθήκη omp είναι αυτά του υπολογισμού των νέων τιμών σε μία γενιά. Κάτι τέτοιο αρχικά είναι εφικτό, επειδή τα συγκεκριμένα κελιά ανανεώνονται ξεχωριστά το ένα από το άλλο σε νέο πίνακα (u2). Από αυτό μπορούμε να καταλάβουμε ότι η χρήση νημάτων μπορεί να προσφέρει παρόμοια βελτίωση στο χρόνο εκτέλεσης του

προγράμματος με αυτή που πρόσφερε και η χρήση πολλών διεργασιών. Οπότε λοιπόν, είναι λογικό να βλέπουμε ο χρόνος των μετρήσεων βελτιώνεται για 4δ-2ν, σε σχέση με τις υπόλοιπες μετρήσεις για 4δ. Επίσης, η επιλογή του συγκεκριμένου συνδυασμού δεν έγινε τυχαία. Όπως βλέπουμε, οι μετρήσεις έγιναν για τα 2 είδη των μικρότερων και μεγαλύτερων πινάκων. Για μικρότερους πίνακες θα είχαμε την ίδια συμπεριφορά με αυτή ων διεργασιών, σε περίπτωση που είχαμε περισσότερα νήματα. Έτσι, παίρνουμε καλύτερους υπολογισμούς για όλα τα ήδη πινάκων. Ενδεχομένως, εάν είχαμε περισσότερα δεδομένα με ακόμα μεγαλύτερες πλευρές, να βόλευε να είχαμε και περισσότερα νήματα.

Σας ευχαριστούμε πολύ.

Γιάννης Ρέππας – 1115201500137

Γιάννης Τασσόπουλος - 1115201500155