

Πανεπιστήμιο Πατρών  
Πολυτεχνική Σχολή  
Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών  
Τομέας Ηλεκτρονικής και Υπολογιστών

Τελική Έκθεση Προόδου  
για την εργασία του μαθήματος  
Προηγμένα Συστήματα Υπολογιστών

Ανάπτυξη παράλληλης εφαρμογής πολλαπλασιασμού  
πινάκων με χρήση κατανεμημένης κοινής μνήμης  
υλοποιημένης σε λογισμικό.

Θωμάς Ρεπαντής  
Έτος: Ε΄  
Κύκλος Σπουδών: Ηλεκτρονική και Υπολογιστές  
Α.Μ.: 4218  
E-mail: darkzero@otenet.gr

Πάτρα, 8.2.2002

**Εισαγωγή.** Σκοπός της εργασίας είναι η ανάπτυξη μιας παράλληλης εφαρμογής πολλαπλασιασμού πινάκων, η οποία θα χρησιμοποιεί κατανεμημένη κοινή μνήμη υλοποιημένη σε λογισμικό (Software Distributed Shared Memory ή SWDSM) και κατόπιν η μελέτη των επιδόσεων αυτής σε σύγκριση και με αντίστοιχη σειριακή εφαρμογή που επίσης αναπτύχθηκε.

## **Χρησιμοποιηθήσα υποδομή για κατανεμημένη κοινή μνήμη.**

**Χαρακτηριστικά** Η παράλληλη εφαρμογή που υλοποιήθηκε χρησιμοποιεί κλήσεις στη βιβλιοθήκη JIAJIA, υλοποιημένης σε C στην Κινέζικη Ακαδημία Επιστημών. Το JIAJIA είναι ένα σύστημα κατανεμημένης κοινής μνήμης υλοποιημένης σε λογισμικό (Software Distributed Shared Memory System) ή αλλιώς εικονικής κοινής μνήμης (Shared Virtual Memory). Υλοποιεί δηλαδή σε λογισμικό την ψευδαίσθηση του ενιαίου εικονικού χώρου διευθύνσεων, που περιλαμβάνει μνήμη από όλους τους κόμβους που συμμετέχουν στην εκτέλεση της εφαρμογής. Το σχήμα της οργάνωσης μνήμης που ακολουθείται είναι ανάλογο της ανομοιόμορφης πρόσβασης στη μνήμη (NUMA (Non Uniform Memory Access) - like), προσφέροντας απλή διαχείριση της κοινής μνήμης. Το προγραμματιστικό μοντέλο που ακολουθείται είναι εκείνο του μοναδικού προγράμματος και πολλαπλών δεδομένων (SPMD (Single Program Multiple Data)).

**API** Το JIAJIA προσφέρει μια διεπαφή για τον προγραμματισμό εφαρμογών API (Application Programming Interface) που περιλαμβάνει συναρτήσεις για βασικές καθώς και για προχωρημένες λειτουργίες. Στις πρώτες συγκαταλέγονται η αρχικοποίηση και η εκκίνηση της εφαρμογής σε όλους τους κόμβους, η κατανομή κοινής μνήμης, η απόκτηση και η απελευθέρωση κλειδιού (lock), η εκτέλεση καθολικού φράγματος και η έξοδος. Οι δεύτερες περιλαμβάνουν ρυθμίσεις της συμπεριφοράς του JIAJIA, την εξισορρόπηση φορτίου, άλλες μεταβλητές συγχρονισμού, την εκτέλεση αναμονής, κλήσης ανάλογες του μηχανισμού διαβίβασης μηνυμάτων (MPI (Message Passing Interface) - like) και την τήρηση στατιστικών στοιχείων.

**Πρακτικές πληροφορίες** Το JIAJIA τρέχει σε αυτοδύναμες μηχανές ή μηχανές που χρησιμοποιούν NFS. Χρησιμοποιεί έναν κόμβο-διαχειριστή και πολλούς κόμβους-εργάτες, όλα οριζόμενα σε ένα αρχείο (.jiahosts). Η εκτέλεση στους κόμβους-εργάτες ξεκινά με rhexec, ενώ στην περίπτωση των αυτο-

δύναμων κόμβων τα αρχεία της εφαρμογής μεταφέρονται με scp. Κάνουμε τις αναγκαίες μετατροπές στον κώδικα, ώστε να χρησιμοποιηθούν τα ασφαλέστερα εναλλακτικά μέσα ssh και scp αντίστοιχα.

## Υλοποίηση

**Γενικά** Η υλοποίηση έγινε σε C. Επιλέξαμε οι πολλαπλασιαστέοι πίνακες να περιέχουν τυχαίους αριθμούς και το μέγεθος τους να ορίζεται στον κώδικα και όχι από το χρήστη. Ο αρχικός έλεγχος της ορθότητας του αλγορίθμου ωστόσο έγινε με μικρά μεγέθη πινάκων και με συγκεκριμένους αριθμούς, ώστε να μπορεί να επαληθευθεί το αποτέλεσμα.

**Παράλληλη εφαρμογή** Η εφαρμογή παράλληλου πολλαπλασιασμού πινάκων που αναπτύχθηκε περιέχει -εκτός από την κύρια συνάρτηση εκτέλεσης- συναρτήσεις αρχικοποίησης των πινάκων και πολλαπλασιασμού τους. Η αρχικοποίηση γίνεται στον κόμβο-διαχειριστή και περιλαμβάνει το γέμισμα των πινάκων με τυχαίους αριθμούς και την αναστροφή του δεύτερου πίνακα, προκειμένου να βελτιστοποιηθεί ο αλγόριθμος του πολλαπλασιασμού εκμεταλλευόμενος την τοπικότητα των δεδομένων. Ο πολλαπλασιασμός σπάει χωρίζοντας τους πίνακες ανάλογα με τον κόμβο στον οποίο βρισκόμαστε. Στην κύρια συνάρτηση περιλαμβάνονται και οι απαραίτητες κλήσεις στο JAJIA για αρχικοποίηση και εκκίνηση της εφαρμογής σε όλους τους κόμβους, κατανομή κοινής μνήμης, εκτέλεση καθολικών φραγμάτων συγχρονισμού, συλλογή στατιστικών δεδομένων, μέτρηση χρόνου και έξοδο.

Υλοποιήσαμε μικρές παραλλαγές της παράλληλης εφαρμογής, διαφοροποιούμενες στο που αποθηκεύονται οι πολλαπλασιαστέοι και το γινόμενο.

1. version 1: Η πρώτη έκδοση κατανέμει τους πολλαπλασιαστέους και το γινόμενο ομοιόμορφα (κυκλικά) σε όλους τους κόμβους, ώστε να ελαττωθεί ο ρυθμός αστοχίας (miss rate). Επιπλέον ο τελικός πίνακας-εσωτερικό γινόμενο παραμένει κομματιασμένος στους κόμβους που υπολόγισαν τα επιμέρους κομμάτια του.
2. version 2: Η δεύτερη έκδοση επίσης κατανέμει κυκλικά τους πίνακες, φέρνει ωστόσο τον τελικό πίνακα-εσωτερικό γινόμενο ολόκληρο στον κόμβο-διαχειριστή.

3. version 3: Η τρίτη έκδοση δεν κατανέμει ομοιόμορφα τους πίνακες, αλλά τους αποθηκεύει όλους στον κόμβο-διαχειριστή. Ο τελικός πίνακας-εσωτερικό γινόμενο δεν επιστρέφεται στον κόμβο-διαχειριστή.
4. version 4: Η τέταρτη έκδοση αποθηκεύει όλους τους πίνακες στον κόμβο-διαχειριστή και επιπλέον επιστρέφει τον τελικό πίνακα-εσωτερικό γινόμενο στον κόμβο- διαχειριστή.

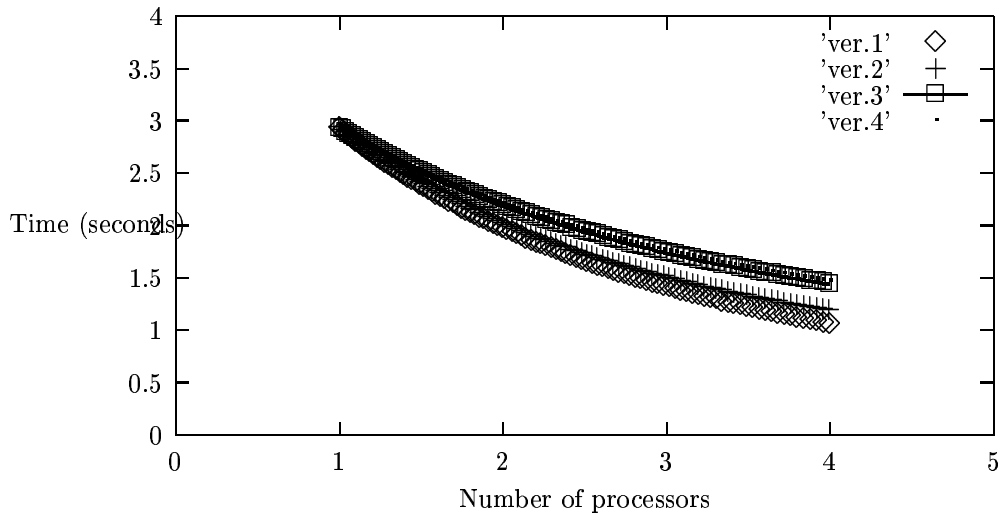
**Σειριακή εφαρμογή** Προκειμένου να συγκρίνουμε τις επιδόσεις της παράλληλης εφαρμογής που αναπτύξαμε, υλοποιήσαμε και μια σειριακή εφαρμογή πολλαπλασιασμού πινάκων, που χρησιμοποιεί τον ίδιο αλγόριθμο, ο οποίος είναι καλός και σε ακολουθιακή εκτέλεση. Για τη μέτρηση χρόνου χρησιμοποιούμε τη συνάρτηση `gettimeofday` που προσφέρει ακρίβεια μικροδευτερολέπτου.

**Μετρήσεις** Οι κυρίως μετρήσεις έγιναν σε συστάδα αποτελούμενη από 4 υπολογιστές με επεξεργαστές Intel Pentium III @ 866MHz (256KB λανθάνουσα μνήμη) και κύρια μνήμη 256MB. Το διασυνδεδετικό δίκτυο ήταν GigaBit Ethernet, το λειτουργικό σύστημα Linux (έκδοση πυρήνα 2.4.16) και ο μεταγλωττιστής `gcc` (έκδοση 2.96). Επιπλέον μετρήσεις με στόχο να εμφανίσουμε χρονοβελτίωση καλύτερη της γραμμικής έγιναν σε συστάδα αποτελούμενη από 2 υπολογιστές με επεξεργαστές Intel Pentium @ 133MHz και κύρια μνήμη 64MB. Το διασυνδεδετικό δίκτυο σε αυτήν την περίπτωση ήταν 100MBit Ethernet, το λειτουργικό σύστημα Linux (έκδοση πυρήνα 2.4.14) και ο μεταγλωττιστής `gcc` (έκδοση 2.95.2). Σε όλες τις περιπτώσεις οι μετρήσεις έγιναν χωρίς επιπλέον κίνηση στο δίκτυο και με ελάχιστο επιπλέον φόρτο στα μηχανήματα. Ακολουθούν οι μετρήσεις με τη μορφή πινάκων και χαρακτηριστικά διαγράμματα. Η χρονοβελτίωση (`speedup`) σε κάθε περίπτωση υπολογίστηκε ως ο λόγος του σειριακού χρόνου εκτέλεσης προς τον παράλληλο, όπου ο σειριακός χρόνος εκτέλεσης ήταν είτε ο χρόνος εκτέλεσης τους ακολουθιακού προγράμματος (`Speedup1`), είτε ο χρόνος εκτέλεσης του παράλληλου προγράμματος (με τις κλήσεις στη βιβλιοθήκη) σε ένα μόνο κόμβο (`Speedup2`) (τιμές που δε διαφέρουν άλλωστε πολύ). Για παράλληλο πολλαπλασιασμό πινάκων 2048x2048 χρησιμοποιώντας την τρίτη και τέταρτη έκδοση του παράλληλου προγράμματος ο συνδυασμός υπολογιστικού φόρτου και φόρτου στο διασυνδεδετικό δίκτυο ήταν τέτοιος που καθυστέρουσε υπερβολικά τη μέτρηση, με αποτέλεσμα να τη διακόψουμε.

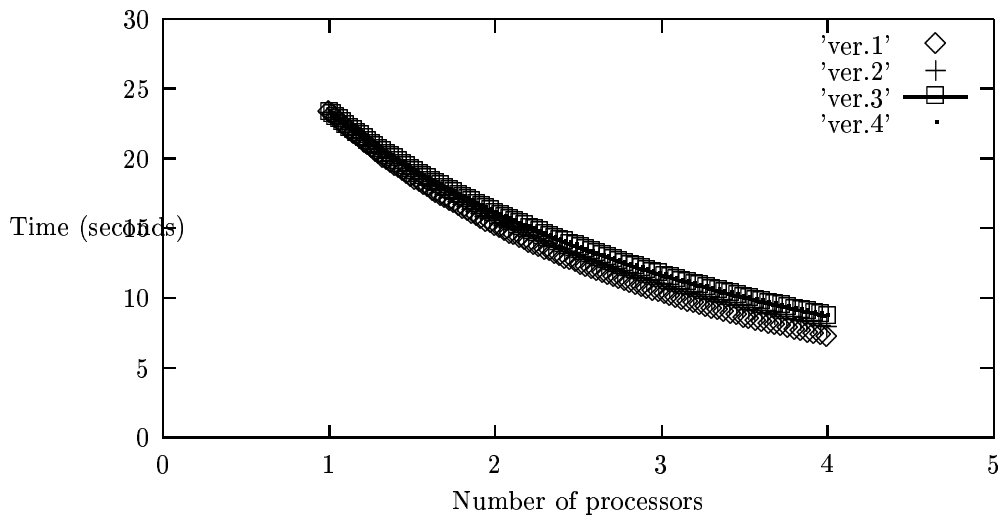
Χρόνοι εκτέλεσης

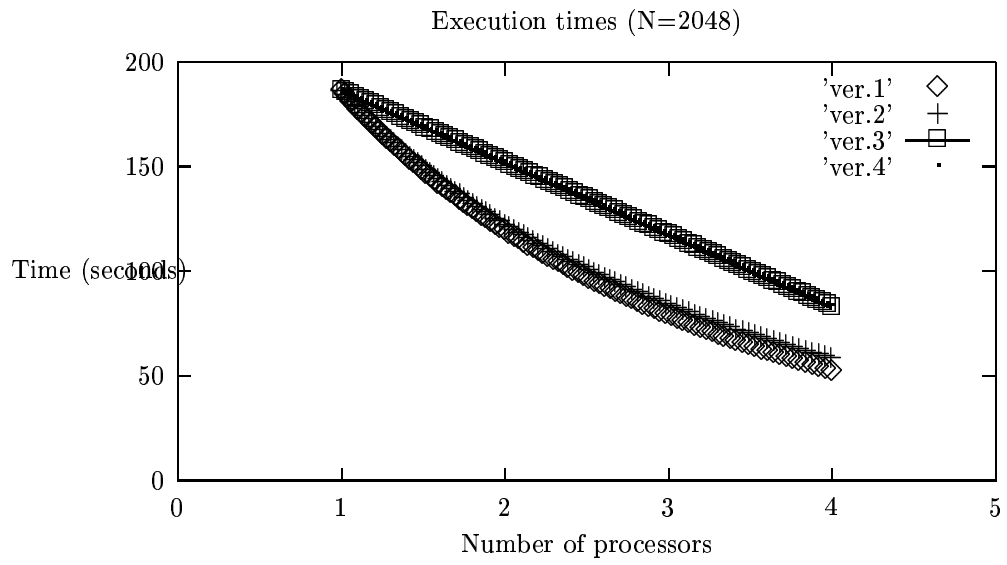
<i>ProblemSize</i>	<i>NumOfProcs</i>	<i>SeqTime(sec)</i>	<i>ParTime(v.1)(sec)</i>	<i>Speedup1</i>	<i>Speedup2</i>
512	1	2.91	2.93	0.993	1.000
	2		1.62	1.796	1.809
	4		1.06	2.745	2.764
			<i>ParTime(v.2)(sec)</i>		
	1		2.95	0.986	1.000
	2		1.68	1.732	1.756
	4		1.20	2.425	2.458
			<i>ParTime(v.3)(sec)</i>		
	1		2.93	0.993	1.000
	2		1.93	1.508	1.518
	4		1.44	2.021	2.035
			<i>ParTime(v.4)(sec)</i>		
	1		2.95	0.986	1.000
	2		1.94	1.5	1.521
	4		1.47	1.980	2.007
			<i>ParTime(v.1)(sec)</i>		
1024	1	23.33	23.30	1.001	1.000
	2		12.15	1.920	1.918
	4		7.18	3.249	3.245
			<i>ParTime(v.2)(sec)</i>		
	1		23.37	0.998	1.000
	2		12.69	1.839	1.842
	4		8.00	2.916	2.921
			<i>ParTime(v.3)(sec)</i>		
	1		23.32	1.000	1.000
	2		13.56	1.721	1.720
	4		8.68	2.688	2.687
			<i>ParTime(v.4)(sec)</i>		
	1		23.36	0.999	1.000
	2		13.60	1.715	1.718
	4		8.72	2.675	2.679
			<i>ParTime(v.1)(sec)</i>		
2048	1	186.65	186.11	1.003	1.000
	2		95.36	1.957	1.952
	4		52.29	3.570	3.559
			<i>ParTime(v.2)(sec)</i>		
	1		186.42	1.001	1.000
	2		100.32	1.861	1.858
	4		58.96	3.166	3.162
		5	<i>ParTime(v.3)(sec)</i>		
	1		186.07	1.003	1.000
	2		<i>n/a</i>	<i>n/a</i>	<i>n/a</i>
	4		82.62	2.259	2.252
			<i>ParTime(v.4)(sec)</i>		
	1		186.41	1.001	1.000
	2		<i>n/a</i>	<i>n/a</i>	<i>n/a</i>

Execution times (N=512)



Execution times (N=1024)

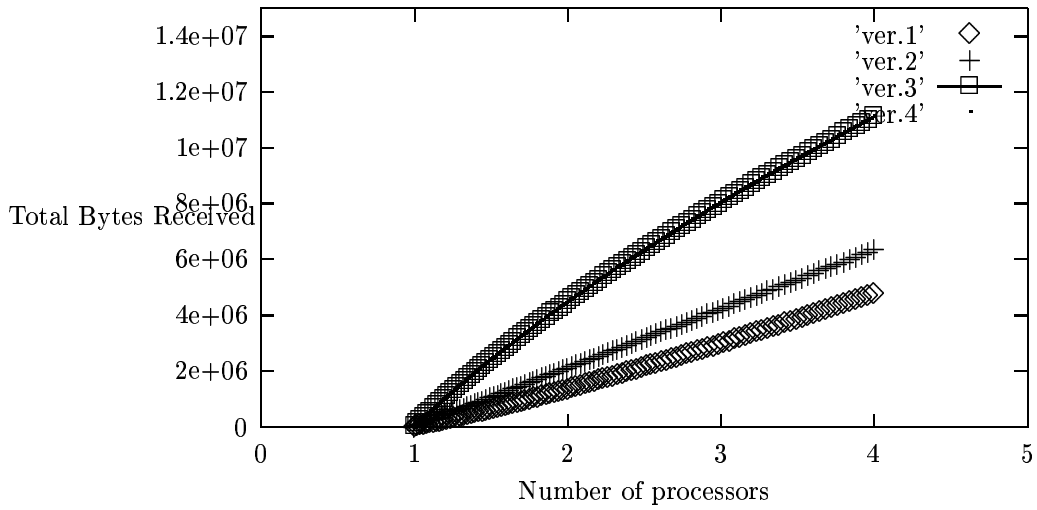




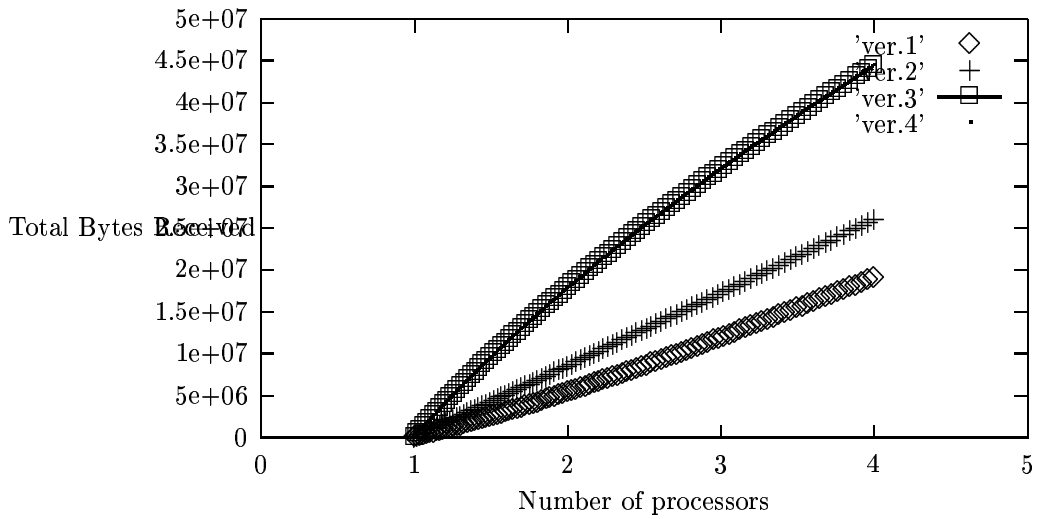
Συνολική πληροφορία που παρέλαβαν οι κόμβοι

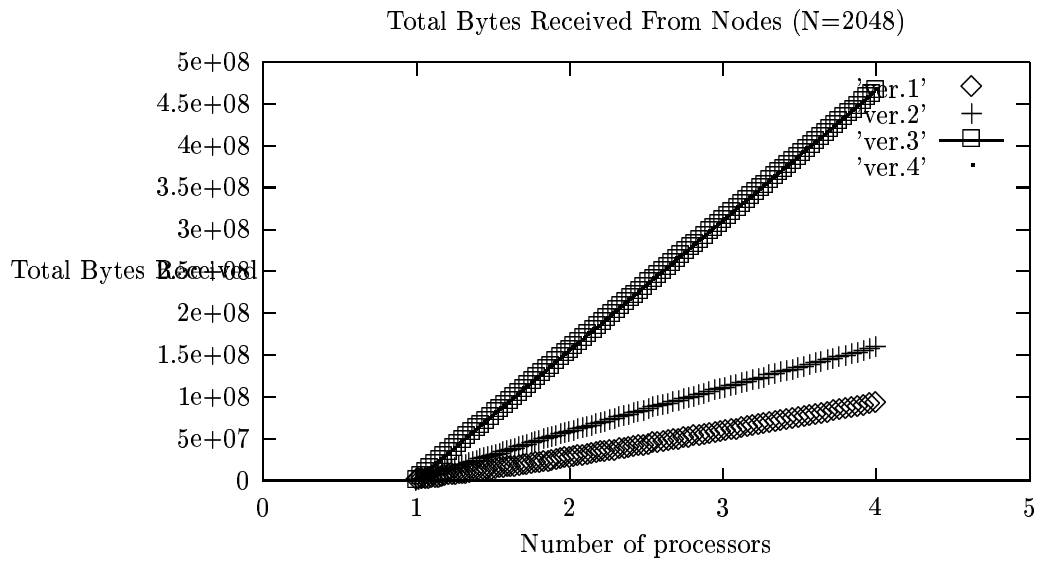
<i>ProblemSize</i>	<i>NumOfProcs</i>	<i>TotalBytesReceived(v.1)</i>
512	1	0
	2	1057892
	4	4760920
		<i>TotalBytesReceived(v.2)</i>
	1	0
	2	2115684
	4	6347608
		<i>TotalBytesReceived(v.3)</i>
	1	0
	2	5285192
	4	11105496
		<i>TotalBytesReceived(v.4)</i>
	1	0
	2	5285192
	4	11105496
		<i>TotalBytesReceived(v.1)</i>
1024	1	0
	2	4231268
	4	19041004
		<i>TotalBytesReceived(v.2)</i>
	1	0
	2	8462436
	4	26073944
		<i>TotalBytesReceived(v.3)</i>
	1	0
	2	21140296
	4	44419032
		<i>TotalBytesReceived(v.4)</i>
	1	0
	2	21140296
	4	44419032
		<i>TotalBytesReceived(v.1)</i>
2048	1	0
	2	22279808
	4	91532704
		<i>TotalBytesReceived(v.2)</i>
	1	0
	2	64422444
	4	159545512
		<i>TotalBytesReceived(v.3)</i>
	1	0
	2	<i>n/a</i>
	4	465657312
		<i>TotalBytesReceived(v.4)</i>
	1	0
	2	<i>n/a</i>

Total Bytes Received From Nodes (N=512)



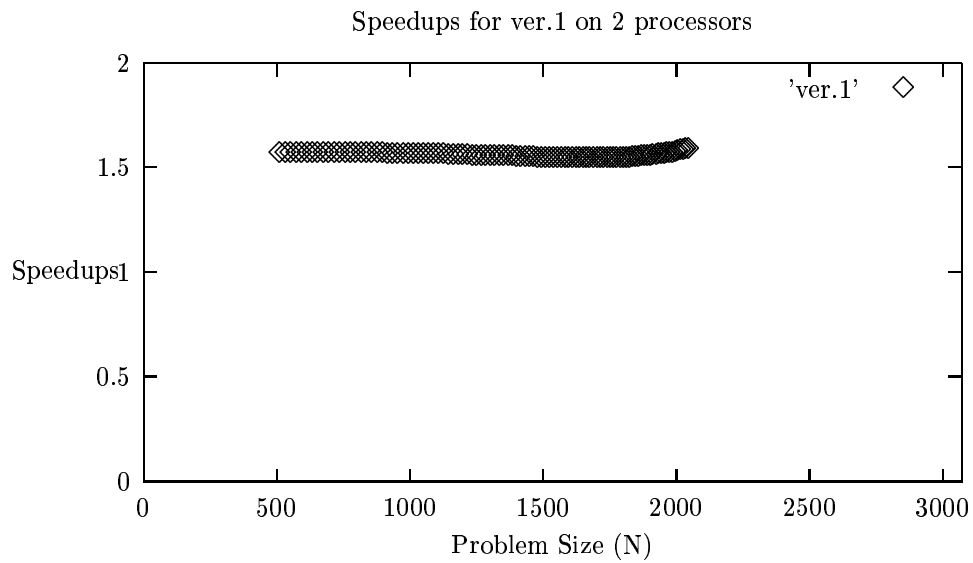
Total Bytes Received From Nodes (N=1024)





Χρονοβελτιώσεις για την έκδοση 1 του παράλληλου προγράμματος σε 2 κόμβους

<i>NumOfProcs</i>	<i>ProblemSize</i>	<i>SeqTime(sec)</i>	<i>ParTime(v.1)(sec)</i>	<i>Speedup</i>
2	512	15.59	9.95	1.567
	1024	146.59	93.36	1.570
	1536	496.58	319.90	1.552
	1800	760.66	506.92	1.501
	2048	1265.89	796.96	1.588



**Συμπεράσματα.** Παρατηρούμε ότι για όλα τα μεγέθη του προβλήματος οι χρόνοι παράλληλης εκτέλεσης είναι αισθητά βελτιωμένοι σε σύγκριση με τους αντίστοιχους σειριακούς. Οι χρονοβελτιώσεις που πετυχαίνονται για 2 κόμβους είναι κοντά στην ιδανική, ενώ και για 4 κόμβους έχουμε χρονοβελτιώσεις αυξημένες κατά 1 ακόμη μονάδα, γεγονός που μας επιτρέπει να εξάγουμε θετικά συμπεράσματα για την κλιμακοσιμότητα του συστήματος στη συγκεκριμένη εφαρμογή. Όπως είναι αναμενόμενο, οι εκδόσεις του παράλληλου προγράμματος με το λιγότερο επιπλέον κόστος για μετακινήσεις πετυχαίνουν και καλύτερους χρόνους εκτέλεσης.

Προφανώς τα bytes που απαιτείται να μεταφερθούν αυξάνονται, καθώς αυξάνεται ο αριθμός των κόμβων που συμμετέχουν στον πολλαπλασιασμό. Από τον ορισμό του εσωτερικού γινομένου των πινάκων  $A \cdot B = C$  και για έναν τεμαχισμένο πολλαπλασιασμό, οι κόμβοι χρειάζονται το  $\frac{1}{NumOfNodes}$  του πίνακα A, ολόκληρο τον πίνακα B και το  $\frac{1}{NumOfNodes}$  του πίνακα C. Συνεπώς, στην καλύτερη περίπτωση της έκδοσης 1, οι κόμβοι χρειάζεται να παραλάβουν τα  $\frac{NumOfNodes-1}{NumOfNodes}$  του B, στην περίπτωση της έκδοσης 2 απαιτείται επιπλέον η επιστροφή  $\frac{NumOfNodes-1}{NumOfNodes}$  του C στον κόμβο-διαχειριστή, στην έκδοση 3 οι κόμβοι-εργάτες χρειάζεται να παραλάβουν το  $\frac{1}{NumOfNodes}$  του πίνακα A, το  $\frac{1}{NumOfNodes}$  του C και ολόκληρο τον B, ενώ στη χειρότερη περίπτωση της έκδοσης 4, απαιτείται επιπλέον η επιστροφή  $\frac{NumOfNodes-1}{NumOfNodes}$  του C στον κόμβο-διαχειριστή. Οφείλουμε να λάβουμε υπ' όψιν μας ότι η αρχικοποίηση των A και B που γίνεται στον κόμβο-διαχειριστή έχει ως αποτέλεσμα να μεταφερθούν οι πίνακες αυτοί ολόκληροι στον κόμβο αυτό, ακόμα και στην περίπτωση που δε βρίσκονται εκεί εξ αρχής. Σημειώνουμε ότι κάθε πίνακας από doubles μεγέθους NxN αποθηκεύεται σε NxNx8bytes. Με βάση τα παραπάνω μπορούμε να ερμηνεύσουμε τα μεγέθη της μετακινούμενης πληροφορίας, που όπως εξηγήσαμε αυξάνουν για τις διαδοχικές περιπτώσεις. Οι μετρήσεις μας έδειξαν ότι δεν παρατηρήθηκε σημαντική διαφορά σε χρόνο και μεταφερόμενα bytes μεταξύ τρίτης και τέταρτης έκδοσης, πιθανώς επειδή το JIAJIA επιστρέφει πριν την έξοδό του τις σελίδες μνήμης στους κόμβους που αυτές ανήκουν (στην περίπτωση μας στον κόμβο-διαχειριστή).

Στη βιβλιογραφία έχουν αναφερθεί περιπτώσεις παράλληλων προγραμμάτων εκτελούμενων σε συστήματα κατανεμημένης κοινής μνήμης που εμφανίζουν χρονοβελτίωση καλύτερη της γραμμικής (superlinear speedup). Το φαινόμενο αυτό οφείλεται στο γεγονός ότι συμβαίνει κάποτε τα δεδομένα να μη χωρούν στην κύρια μνήμη ενός κόμβου, με αποτέλεσμα ο σειριακός αλγόριθμος να οδηγήσει σε συνεχές swapping στο σκληρό δίσκο (λόγω χρήσης της εικονικής μνήμης) και τελικά στο λυγισμό (thrashing), ενώ ο παράλληλος αλγόριθμος που

δεν απαιτεί τη συγκέντρωση όλων των δεδομένων σε έναν κόμβο δε συναντά αντίστοιχο πρόβλημα (αποφεύγει δηλαδή τις αργές προσβάσεις στο σκληρό δίσκο και χρησιμοποιεί γρήγορες προσβάσεις μέσω του δικτύου στην κύρια μνήμη άλλων κόμβων). Πήραμε επιπλέον μετρήσεις σε συστάδα 2 κόμβων με μικρή κύρια μνήμη με στόχο να εμφανίσουμε χρονοβελτίωση καλύτερη της γραμμικής. Δυστυχώς κάτι τέτοιο δεν επιτεύχθηκε για διάφορους λόγους: Το αργό διασυνδεδετικό δίκτυο εισήγαγε επιπλέον καθυστέρηση στον παράλληλο πολλαπλασιασμό. Επιπλέον ο αλγόριθμος του πολλαπλασιασμού που υλοποιήσαμε προσπαθεί να εκμεταλλευθεί την τοπικότητα των δεδομένων στους κόμβους, με αποτέλεσμα όμως να εκμεταλλεύεται και την τοπικότητα των δεδομένων στην κύρια μνήμη στην περίπτωση και της ακολουθιακής εκτέλεσης και να αποφεύγει λυγισμό μέχρι κατάρρευσης. Τέλος όπως εξηγήσαμε απαιτείται αρκετή αποθήκευση πληροφορίας ακόμα και στην περίπτωση του παράλληλου αλγορίθμου, ιδιαίτερα στον κόμβο-διαχειριστή που αναλαμβάνει και τις αρχικοποιήσεις των πινάκων. Ίσως να μπορούσαμε να παρατηρήσουμε το φαινόμενο σε πολλαπλασιασμό με αρχικοποιήσεις που γίνονταν τοπικά. Πιθανώς να βοηθούσε και ένα πιο γρήγορο διασυνδεδετικό δίκτυο ή και ένας άλλος αλγόριθμος πολλαπλασιασμού. Πάντως είχαμε την ευκαιρία -ακόμη και υπό αυτές τις συνθήκες- να παρατηρήσουμε την έντονη χρήση του σκληρού δίσκου από το συνεχές swarming.

**Συνημμένα.** Επισυνάπτεται ο πηγαίος κώδικας της σειριακής και των παράλληλων εφαρμογών σε μορφή κειμένου. Επίσης σε δισκέττα επισυνάπτονται το παρόν κείμενο σε ηλεκτρονική μορφή, οι έξοδοι από τις εκτελέσεις των προγραμμάτων και οι αντίστοιχες μετρήσεις, τα Makefiles, καθώς και όλοι οι πηγαίοι κώδικες και τα αντίστοιχα εκτελέσιμα (και για τη βιβλιοθήκη).