Πανεπιστήμιο Πατρών
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής
Τομέας Λογισμικού των Υπολογιστών

Τελική Έκθεση
για την εργαστηριακή άσκηση 2 του μαθήματος
Λογισμικό και Προγραμματισμός Συστημάτων Υψηλής
Επίδοσης

Θωμάς Ρεπαντής
Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών
Κύκλος Σπουδών Ηλεκτρονικής και Υπολογιστών
Έτος: Ε'
Α.Μ.: 4218
Ομάδα: hpc6
E-mail: darkzero@otenet.gr

Πάτρα, 22.3.2002
Εισαγωγή. Σκοπός της εργασίας είναι η υλοποίηση και ο έλεγχος της σωστής λειτουργίας βιβλιοθήκης χρόνου εκτέλεσης για τη διαχείριση νημάτων (threads). Η βιβλιοθήκη, που ονομάζεται pthreads (parallel systems threads, περιλαμβάνει βασικές συναρτήσεις διαχείρισης νημάτων και βασικούς αλγορίθμους συγχρονισμού. Τα νήματα είναι επιπέδου χρήστη (user-level threads) και (non-preemptive). Τα νήματα επιπέδου χρήστη έχουν μικρότερο χόστος διαχείρισης από αυτά επιπέδου πυρήνα, αφού δεν απαιτούν τη μετάβαση του επεξεργαστή από το επίπεδο επιπέδου χρήστη στο επίπεδο επιπέδου πυρήνα. Επιτρέπουν έτσι την εκμετάλλευση λεπτά καταμερισμένου παραλληλισμού εφαρμογών. Κάθε παράλληλη εφαρμογή προκειμένου να είναι αποδοτική οφείλει να χρησιμοποιεί μια υβριδική τεχνική, κατά την οποία δημιουργεί πλήθος νημάτων επιπέδου πυρήνα ίσο με τον προγραμματικό ευκταίο παραλληλισμό και τα οποία ονομάζουμε ιδιαίτερα επεξεργαστές, οι οποίοι εκτελούν τα νήματα επιπέδου χρήστη που ο λεπτά καταμερισμένος παραλληλισμός επιβάλλει.

Υλοποίηση βιβλιοθήκης διαχείρισης νημάτων. Η βιβλιοθήκη που υλοποιήθηκε σε C περιλαμβάνει τους τύπους δεδομένων και τις συναρτήσεις που όρισε το σύστημα διεπαφής με τον προγραμματιστή (Application Programming Interface) των προδιαγραφών, καθώς και τύπους δεδομένων και συναρτήσεις για εσωτερική χρήση. Αναλυτικά αυτά είναι:

Τύποι δεδομένων του API,

- pthread_t για νήμα επιπέδου χρήστη. Στον περιγραφέα αυτού συμπεριλάβαμε δείκτη στη στοιχείου του νήματος, δείκτη στο προσγευμένο και στο επίμενο νήμα στην ουρά, χώρο για να αποθηκεύεται το περιβάλλον του νήματος, έναν αχεραίο ο οποίος εκφράζει την κατάσταση του νήματος (εκτελούμενο, έτοιμο προς εκτέλεση, zombie) (ο οποίος ίσως μπορούσε να αποφευχθεί, αφού το νήμα ανάλογα με την κατάστασή του βρίσκεται ήδη στην αντίστοιχη ουρά -εφόσον όλα ταίνιον καλώς), δείκτη στη συνάρτηση που εκτελείται κατά την εκκίνηση του νήματος, δείκτη στα ορίσματα αυτής, δείκτη στο νήμα που έχει χάνει join στο νήμα αυτό (εάν έχει σύμβει κάτι τέτοιο) (εξαπληρείτε στην αλληλεπίδραση των pthread_exit(), pthread_join()) και lock που συγχρονίζει τις αλλαγές στα πεδία του περιγραφέα και που πάλι εξαπληρεί την αλληλεπίδραση των pthread_exit(), pthread_join(), αφού εξασφαλίζει ότι η pthread_join() δε θα ελευθερώσει την περιοχή μήνυμας που αντιστοιχεί στον περιγραφέα του νήματος πριν η pthread_exit() για το αντίστοιχο νήμα δεν αλληλεπιδράσει την εκτέλεσή της, κάτι που θα προκαλούσε προβλήματα.
• pthread_lock_t για μεταβλητή συγχρονισμού. Χρησιμοποιήθηκε το Test and Test and Set Lock, volatile για λόγους ορθότητας και padded για λόγους επίδοσης.

• pthread_barrier_t για φράγμα. Χρησιμοποιήθηκε το Sense Reversing Centralized Incremental Barrier, volatile και padded για τους ίδιους λόγους.

Συναρτήσεις του API.

• int pthread_init(int kernel_threads) για την αρχικοποίηση των εσωτερικών δομών της βιβλιοθήκης και για τη δημιουργία του ζητούμενου αριθμού νήματων επιπέδου πυρήνα (ιδιαίτερα επεξεργαστών). Τα νήματα επιπέδου πυρήνα δημιουργήθηκαν με τις χλήσεις που ορίζει το πρότυπο POSIX threads για λόγους μεταφερσιμότητας, αλλά και για ευκολότερη δοκιμαστικά και δημιουργία τους με τη βοήθεια της χλήσης συστήματος thr_create() του Solaris. Στα νήματα αυτά δόθηκαν σε κάθε περίπτωση οι κατάλληλες ιδιότητες, ώστε να είναι bound (scheduled on a system-wide basis). Δημιουργούμε επίσης ένα υποτρόπιο νήμα επιπέδου χρήστη, αντίστοιχο του ήδη εκτελούμενου χώδικα, το οποίο χρησιμοποιείται μόνο για το πρώτο context-switch και δεν τοποθετείται σε κάποια ουρά. Ακόμα δημιουργούμε άργο νήμα για τον ιδιαίτερο επεξεργαστή ζάλι που συνιστάται όταν δεν υπάρχουν χρήσεις νήματα επιπέδου χρήστη προς εκτέλεσή.

• int pthread_create(pthread_t **thread_desc, void (start_routine)(void *), void *arg, int queue) για τη δημιουργία ενός νήματος επιπέδου χρήστη

• void pthread_exit(void) για τον τερματισμό της εκτέλεσης ενός νήματος επιπέδου χρήστη, χωρίς όμως την απελευθέρωση των αντίστοιχων εσωτερικών δομών.

• int pthread_join(pthread_t *thread_desc) για την αναστολή της εκτέλεσης ενός νήματος επιπέδου χρήστη έως ότου τερματίσει κάποιο άλλο συγκεκριμένο, ήπειρο και απελευθέρωνονται οι αντίστοιχες εσωτερικές δομές του τερματιστήρα.

• void pthread_yield(void) για την παραχώρηση του επεξεργαστή από ένα νήμα στη βιβλιοθήκη, η οποία αναλαμβάνει την εκτέλεση άλλου νήματος. Η εκτέλεση του παραχωρήσαντος συνεχίζεται αργότερα από το σημείο που σταμάτησε. Προκειμένου να αποφευχθεί η περίπτωση άργοι
επεξεργαστές να ξεκινήσουν την εκτέλεση νήματος που καλεί την `psthread_yield()` πριν η εκτέλεση αυτής ολοκληρωθεί ή πρέπει να ληφθεί ειδική πρόνοια εκκαθάρισης του περιβάλλοντος στην `MD_LONGJMP()` ή η `ps_schedule()` να εκτελείται ως εξοχοποιημένο νήμα που θα διασφαλίζει το συγχρονισμό αυτό.

- `psthread.t` `psthread_self(void)` για την εύρεση του περιγραφέα του νήματος που την καλεί.
- `int psthread_lock_init(psthread_lock_t *lock_var)` για την αρχικοποίηση μεταβλητής συγχρονισμού.
- `int psthread_get_lock(psthread_lock_t *lock_var)` για τη λήψη της κυριότητας μεταβλητής συγχρονισμού. Για την απομονωμένη αδιεξοδική (μιας και τα νήματα είναι μη προεκτοποιημένα) η ενεργός αναμονή έχει αντικατασταθεί από άμεση απελευθέρωση του επεξεργαστή μέσω της `psthread_yield()`.
- `int psthread_release_lock(psthread_lock_t *lock_var)` για την απελευθέρωση μεταβλητής συγχρονισμού.
- `int psthread_barrier_init(pthread_barrier_t *barrier, int thread_count)` για την αρχικοποίηση φράγματος.
- `int psthread_barrier_wait(pthread_barrier_t *barrier)` για την εκτέλεση φράγματος. Ομοίως χρησιμοποιείται πολιτική άμεσης απελευθέρωσης του επεξεργαστή.

Τύποι δεδομένων για εσωτερική χρήση.

- `int NUM_OF_VPs` για τον αριθμό των ιδιαίτερων επεξεργαστών (νημάτων επιπέδου πυρήνα) που θα χρησιμοποιηθούν, όπως τον αφίζει η κλήση της `psthread_init()`.
- `ps_queue_t` για ουρές, υλοποιημένες ως κυκλικές διπλά- διασυνδεδεμένες λίστες, προστατευόμενες από `lock`.
- `ps_queue_t ready_queue` για την ουρά των νημάτων επιπέδου χρήστη που είναι έτοιμη προς εκτέλεση. Χρησιμοποιήθηκε μια καθολική ουρά.
- `ps_queue_t zombie_queue` για την ουρά των νημάτων επιπέδου χρήστη που έχουν τερματίσει και περιμένουν να τους συμβεί `join` από κάποιο άλλο νήμα.
• ps_stack_t για τη στοιβα ενός νήματος επιπέδου χρήστη (παρεχόταν έτοιμη). Το μέγεθος κάθε στοιβας τέθηκε στα 16kB.

• psthread_t *ps_klt[NUM_OF_VPs] για την αποθήκευση του νήματος επιπέδου πυρήνα για κάθε ιδεατό επεξεργαστή. Χρησιμοποιείται για παράδειγμα από την .ps_schedule() και την psthread_self().

• psthread_t *ps_idle_thread[NUM_OF_VPs] για να δείχνει το άργο νήμα επιπέδου χρήστη για κάθε ιδεατό επεξεργαστή. Χρησιμοποιείται για παράδειγμα από την .ps_schedule() και την psthread_self().

• psthread_t *ps_active_thread[NUM_OF_VPs] για να δείχνει το κάθε στιγμή ενεργό νήμα επιπέδου χρήστη για κάθε ιδεατό επεξεργαστή. Χρησιμοποιείται για παράδειγμα από την .ps_schedule() και την psthread_self().

• volatile long int *ps_acttthread_counter για να αποθηκεύεται ο συνολικός αριθμός των εκάστοτε ενεργών νημάτων επιπέδου χρήστη. Δηλώνεται ως volatile, αφού αλλάζει από πολλά νήματα σε διαφορετικούς επεξεργαστές. Χρησιμοποιείται για να διαπιστώσει η ολοκλήρωση και ο τερματισμός της εφαρμογής, από την .ps_idle().

• psthread_lock_t *mutex_counter για την προστασία του παραπάνω μετρητή.

Συναρτήσεις για εσωτερική χρήση.

• void _ps_run(void) για την εκτέλεση της συνάρτησης εκκίνησης ενός νήματος επιπέδου χρήστη με τα ορισμάτα της, αφού η MD_INIT_CONTEXT() δεχεται ως όρισμα συνάρτησης εκκίνησης χωρίς ορισμάτα (οπότε και της παράδειγμα την _ps_run(oid)).

• int _ps_find_vnum(void) για την έναρξη του ιδεατού επεξεργαστή στον οποίο εκτελούμαστε, στοιχείο απαραίτητο για την έναρξη του αντίστοιχου ενεργού ή άργου νήματος.

• void _ps_idle(void) για την απασχόληση του επεξεργαστή όσο δεν υπάρχει χρήση εργασία και η εφαρμογή δεν έχει τερματισεί.

• void _ps_schedule(void) για τη χρονοδιακοπή, τη μετάβαση του επεξεργαστή από ένα νήμα επιπέδου χρήστη σε άλλο.
• void *ps_vp_init(void *vnum) για την αρχικοποίησή των ιδεατών επεξεργαστών. Περιλαμβάνει εξανά τη δημιουργία υποτυπώδους νήματος επιπέδου χρήστη, αντίστοιχου του ήδη εκτελούμενου χώδικα, το οποίο χρησιμοποιείται μόνο για το πρώτο context-switch και δεν τοποθετείται σε κάποια ουρά και τη δημιουργία άνεργου νήματος.

• _inline_ char test_and_set (volatile char *mem) για έλεγχο και ανάθεση τιμής σε μεταβλητή ατομικά (υλοποιημένη σε assembly) (παρεχόταν έτοιμη).

• _inline_ int fetch_and_add (volatile long *mem, long val) για έλεγχο και πρόσθεση τιμής σε μεταβλητή ατομικά (υλοποιημένη σε assembly) (παρεχόταν έτοιμη).

• MD_SETJMP(env) σώζει το context ενός νήματος (περιβάλλον, δηλαδή τιμές καταχωρητών) και χρησιμοποιείται για τη μετάβαση του επεξεργαστή από ένα νήμα σε άλλο (context-switch) (παρεχόταν έτοιμη).

• MD_LONGJMP(env) αποκαθιστά το context ενός νήματος και επίσης χρησιμοποιείται για τη μετάβαση του επεξεργαστή από ένα νήμα σε άλλο (context-switch) (παρεχόταν έτοιμη).

• MD_INIT_CONTEXT(_thread, _sp, _main) αρχικοποιεί το context (τιμές instruction pointer, stack pointer) ενός νήματος που θα χρησιμοποιηθεί για τη μετάβαση του επεξεργαστή από ένα νήμα σε άλλο (context-switch) (παρεχόταν έτοιμη).

• ps_stack_t *ps_stack_new(void) για τη δέσμευση και αρχικοποίηση μιας νέας σταθμών.

• void _ps_lock_init(pthread_mutex_t *thr_ttas_lock_ptr) για την αρχικοποίηση εσωτερικής μεταβλητής συγχρονισμού.

• void _ps_lock(pthread_mutex_t *thr_ttas_lock_ptr) τη λήψη της κυρίτητας εσωτερικής μεταβλητής συγχρονισμού. Χρησιμοποιείται ενεργός αναμονή (διαδοχικές εξετάσεις) - (polling).

• void _ps_unlock(pthread_mutex_t *thr_ttas_lock_ptr) για την απελευθέρωση εσωτερικής μεταβλητής συγχρονισμού.

• void _ps_barrier_init(pthread_barrier_t *thr_incr_barrier, int processors) για την αρχικοποίηση εσωτερικού φράγματος.
- void ps_barrier(pthread_barrierr_t *thr_incr_barrier) για την εκτέλεση εσωτερικού φράγματος. Ομοίως χρησιμοποιείται πολιτική διαδοχικών εξετάσεων - ενεργού αναμονής.

- void ps_queue_init(ps_queue_t *q) για την αρχικοποίηση ουράς.

- void ps_queue_insert(ps_queue_t *q, pthread_t *t) για την εισαγωγή στοιχείου στην κορυφή ουράς.

- void ps_queue_append(ps_queue_t *q, pthread_t *t) για την εισαγωγή στοιχείου στο τέλος ουράς.

- pthread_t *ps_queue_removelfirst(ps_queue_t *q) για τη λήψη στοιχείου από την κορυφή ουράς.

- pthread_t *ps_queue_remove(ps_queue_t *q, pthread_t *t) για την εξαγωγή συγκεκριμένου στοιχείου (ανεξάρτητα από τη θέση του) από ουρά.

Χρησιμοποιώντας την args (αντίστοιχη των ar g και ranlib) δημιουργήσαμε τη στατική βιβλιοθήκη χρόνου εκτέλεσης libsync.a. Προκειμένου να χρησιμοποιήσουν οι υπηρεσίες που προσφέρει από άλλα προγράμματα δημιουργήσαμε και το αντίστοιχο header file (libpthreads.h).

Τύποποιηση πολυνηματικών προγραμμάτων.

Τύποποιηση βοηθητικών προγραμμάτων ελέγχου. Προκειμένου να ελέγχουμε τη σωστή λειτουργία της βιβλιοθήκης αναπτύσσουμε κατά τη διάρκεια της υλοποίησης μικρά προγράμματα που χρησιμοποιούσαν τα αντίστοιχα τμήματα της βιβλιοθήκης. Στα προγράμματα αυτά συμπεριλαμβάναμε κλήσεις για τις νέες συναρτήσεις της βιβλιοθήκης που υλοποιούμε. Συμπεριλαμβάνουμε τις τελικές μορφές των δύο βασικών προγραμμάτων ελέγχου.

Το testqueue καλεί τις εσωτερικές συναρτήσεις της βιβλιοθήκης που έχουν να κάνουν με τη διαχείριση ουρών και αλλάζει έτσι συνεχώς τη μορφή μιας ουράς. Περιλαμβάνει συνάρτηση εκτύπωσης των περιεχομένων ουράς που μας επιτρέπει να δούμε αν οι άλλαγές που επέφεραν οι κλήσεις των συναρτήσεων της βιβλιοθήκης στην ουρά ήταν ήνως οι επιδιούμενες.

Το testlib καλεί τις συναρτήσεις του API της βιβλιοθήκης και βασικές εσωτερικές συναρτήσεις, όπως η ps_find_Vnum(). Το πρόγραμμα αρχικοποιεί τη
βιβλιοθήκη, δημιουργεί νήματα επιπέδου χρήσης τα οποία αυτοπροσδιορίζονται,
αυξάνον ένα μετρητή και βγάζουν και κατόπιν τα κάνει jjoin. Ο έλεγχος του
μετρητή και η εκτύπωση μηνυμάτων μας επιτρέπουν να δούμε αν όλα τα νή-
ματα εκτελέστηκαν σωστά. Αρχικά δοκιμάσαμε τη βιβλιοθήκη σε έναν ιδιαίτερα
επεξεργαστή και στη συνέχεια σε περισσότερους. Προσεκίνησε και ο ιδιαίτερος
επεξεργαστής 0 να μπορεί να εκτελεί νήματα επιπέδου χρήσης, καλεί την
pthread_yield() ή την pthread_join() μετά την αρχικοποιήση της βιβλιοθήκης και
tη δημιουργία νημάτων επιπέδου χρήσης για να μπει στο βρόχο απόσπασης
νημάτων από τη ready-queue.

Τλοποίηση εφαρμογής πολλαπλασιασμού πινάκων. Προχειμένου
να ελεγξούμε την αποδοτικότητα και τη λειτουργικότητα της βιβλιοθήκης υλο-
ποιούμε εφαρμογή (matmul.c) πολλαπλασιασμού (υπολογισμού του εσωτερικού
γγονομένου) δύο πινάκων αριθμών χινητής υποδιαστολής διπλής ακρίβειας, με-
γέδης 2048x2048, αρχικοποιημένως με τυχαίους αριθμούς. Το πρόγραμμα
dέχεται ως είσοδο τον αριθμό των ιδεατών επεξεργαστών και των αριθμών
ημάτων επιπέδου χρήσης και την εντολή να εκτελέσει ή όχι τον πολλαπλασιασμό και
σειριακά και να επαληθεύει το παράλληλο αποτέλεσμα (σύμπτωση σε 3 δεκαδι-
κά ψηφία). Χρησιμοποιούμε τον ίδιο αλγόριθμο πολλαπλασιασμού πινάκων για
tον παράλληλο και σειριακό πολλαπλασιασμό, ο οποίος χρησιμοποιώντας την
tοπικότητά της μνήμη επιτρέπει αυξημένη επίδοση. Στην περίπτωση του παράλλη-
ληλού πολλαπλασιασμού χάνει νόμα επιπέδου χρήσης αναλαμβάνει να υπολογί-
σει ένα τμήμα του πίνακα-αποτελέσματος. Οι κριτήριες χρόνου περιλαμβάνουν
στην περίπτωση του παράλληλου πολλαπλασιασμού το χρόνο αρχικοποίησης
tης βιβλιοθήκης, το χρόνο εκτέλεσης των νημάτων και το χρόνο τερματισμού
και απελευθέρωσης των δομών τους.

Makefile, ορίσματα γραμμής εντολών. Προχειμένου να αποτοποιήσουμε
tη διαδικασία δημιουργίας της βιβλιοθήκης και των εφαρμογών και να στο-
φεύγουμε άσκοπες μεταγλωττίσεις πηγαίου κώδικα που δεν έχει τροποποιηθεί
χρησιμοποιήσαμε το make και γράφαμε το κατάλληλο Makefile. Εξει φαίνονται
και οι παράμετροι που περάσαμε στο μεταγλωττιστή (gcc): -DREENTRANT
-D_SMP_ (για το threading) -O6 (για τη μέγιστη βελτιστοποίηση, ώστε να
μειώσουμε το χρόνο εκτέλεσης) και προαιρετικά -Wall για να βλέπουμε όλες τις
προειδοποιήσεις και στο συνέδετη (gcc): -L$(LIB_PATH) (για τον εντοπισμό
της βιβλιοθήκης που υλοποιήσαμε) -lpthread (για σύνδεση με τη βιβλιοθήκη
POSIX νημάτων) -lpthreads (για σύνδεση με τη βιβλιοθήκη νημάτων που υλο-
ποιήσαμε) για τη δημιουργία των εφαρμογών. Αντίστοιχα κατά τη μεταγλώ-
τιση της βιβλιοθήκης περνάμε τις συναλλαγές παραμέτρους στο μεταγλωττιστή:
-mcpu=ultrasparc -O6 και προαρρητικά τη -Wall. Είναι φανερό ότι πρέπει να
dηλώσουμε τον επεξεργαστή του οποίου την assembly χρησιμοποιούμε στις
συναρτήσεις υλοποίησης των ατομικών πράξεων.

Μετρήσεις. Οι μετρήσεις -όπως και η υλοποίηση- πραγματοποιήθηκαν σε
συμμετρικό πολυεπεξεργαστικό σύστημα κοινής μνήμης με 4 επεξεργαστές Ul-
traSparcII @ 400MHz και λειτουργικό σύστημα Solaris 7 (galois.ceid.upatras.gr).
Δυστυχώς το σύστημα χρησιμοποιούνταν και από άλλους χρήστες (2) κατά τη
διάρκεια των μετρήσεων. Προσπαθήσαμε όμως να εκτελούμε τις μετρήσεις σε
χρονικές στιγμές που το φόρτο του συστήματος -όπως δίνεται από την urtime-
ήταν χαμηλός, (0.5). Πειραματιστήκαμε με τον αριθμό των ιδεατών επεξερ-
γαστών για να υπολογίσουμε τη χρονοβολήση που επιτυγχάνεται και με τον
αριθμό των νημάτων επιπέδου χρήσης για να υπολογίσουμε τη βελτιστοποίηση
του χρόνου εκτέλεσης. (Δίνοντας ps -L κατά τη διάρκεια του χρόνου εκτέλεσης
είδαμε τα νήματα που δημιουργήσαμε και τα επιπέδη νήματα ελέγχου που κατα-
σκευάζει αυτόματα η βιβλιοθήκη νημάτων χρόνου εκτέλεσης του λειτουργικού
συστήματος και τα οποία δεν παρέμειναν επεξεργαστικό χρόνο.) Ακολουθούν οι
μετρήσεις υπό τη μορφή πινάκων και διαγραμμάτων και σχολιασμός - ερμηνεία
tων αποτελεσμάτων:
Χρόνος εκτέλεσης για μεταβλητό αριθμό εικοσικών επεξεργαστών. Στην περίπτωσή αυτή μεταβάλλει τον αριθμό των εικοσικών επεξεργαστών από 1 έως τον αριθμό των φυσικών επεξεργαστών. Ορίσαμε τον αριθμό των νημάτων επιπέδου χρήσης ισο με τον αριθμό των νημάτων επιπέδου πυρήνα (ιδεατών επεξεργαστών). Μετρήσαμε το χρόνο παράλληλης εκτέλεσης.

<table>
<thead>
<tr>
<th>NumOfVPs (= k ips = ul ts)</th>
<th>ExecutionTime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>951.49</td>
</tr>
<tr>
<td>2</td>
<td>482.22</td>
</tr>
<tr>
<td>4</td>
<td>249.22</td>
</tr>
</tbody>
</table>

Execution Times
Χρονοβελτίωση για μεταβλητό αριθμό εικονικών επεξεργαστών.
Ομοίως με την παραπάνω περίπτωση μεταβάλλαμε τον αριθμό των εικονικών
επεξεργαστών και μετρούσαμε χρόνους παράλληλης και σειριακής εκτέλεσης
υπολογίζοντας τη χρονοβελτίωση (speedup) για κάθε περίπτωση.

<table>
<thead>
<tr>
<th>NumOfVPs (= klts = ults)</th>
<th>SeqTime (sec)</th>
<th>ParTime (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>951.13</td>
<td>951.49</td>
<td>0.999</td>
</tr>
<tr>
<td>2</td>
<td>952.86</td>
<td>482.22</td>
<td>1.976</td>
</tr>
<tr>
<td>4</td>
<td>1047.78</td>
<td>249.22</td>
<td>4.204</td>
</tr>
</tbody>
</table>

![Speedups Graph](image-url)
Χρόνος εκτέλεσης για σταθερό αριθμό εικονικών επεξεργαστών (4) και μεταβλητό αριθμό νημάτων επιπέδου χρήσης. Στην περίπτωση αυτή μεταβάλλεται τον αριθμό των νημάτων επιπέδου χρήσης, διατηρώντας σταθερό τον αριθμό των ιδεατών επεξεργαστών ίσο με τον αριθμό των φυσικών επεξεργαστών (4). Ξεκινήσαμε από τόσα νήματα όσα και ο βαθμός παράλληλισμού (4) και προσπαθήσαμε να φθάσουμε στη χρήση ενός νήματος για τον υπολογισμό του κάθε στοιχείου του πίνακα αποτελέσματος. Φυσικά περιορισμοί στη μνήμη του συστήματος δε μας επέτρεψαν τελικά τη δημιουργία 2048x2048 νημάτων επιπέδου χρήσης (για την ακρίβεια ο επιτρεπτός αριθμός ήταν μικρότερος των 100000). Καταφέραμε πάντως να φθάσουμε σε έναν ικανοποιητικό αριθμό και μετρήσαμε σε κάθε περίπτωση το χρόνο παράλληλης εκτέλεσης.
Συμπεράσματα. Παρατηρούμε ότι ο χρόνος εκτέλεσης μειώνεται σχεδόν γραμμικά καθώς αυξάνεται ο αριθμός των ιδεατών επεξεργαστών. Αυτό είναι αναμενόμενο, μιας και κάθε επεξεργαστής μπορεί να υπολογίζει ανεξάρτητα ένα τμήμα του πίνακα αποτελέσματος. Αυτή η έλλειψη ανάγκης συγχρονισμού εξηγεί και τις καλές χρονοβολιώσεις που επιτυγχάνονται. Ο χρόνος σειριακής εκτέλεσης παρατηρούμε ότι μεταβάλλεται -αν και ελάχιστα- στις εκτέλεσεις του προγράμματος με αριθμό ιδεατών επεξεργαστών μεγαλύτερο του 1, πιθανώς μέσα στα πλαίσια της πευκοματικής ακρίβειας. Έτσι και η υπεργραμμική χρονοβολιώση που παρατηρούμε για 4 ιδεατούς επεξεργαστές πιθανά να είναι τυχαία, αν και δεν αποκλείεται να οφείλεται σε χαμηλότερη της τριπλότητας της μηνής ή σε βελτιστοποιήσεις του μεταγλωττισμού. Σχετικά με το μεταβλητό αριθμό νημάτων επιπέδου χρήσης στην περίπτωση που χρησιμοποιούμε ιδεατούς επεξεργαστές όσως και φυσικούς, μπορούμε να παρατηρήσουμε ότι μέχρι έναν αριθμό νημάτων επιπέδου χρήσης (64) έχουμε μείωση του χρόνου εκτέλεσης, μιας και εκμεταλλευόμαστε το λεπτά καταμερισμένο παραλληλισμό της εφαρμογής. Κατάπιν το κόστος διαχείρισης του μεγάλου αριθμού νημάτων επιπέδου χρήσης (αν και μικρότερο από αυτό της διαχείρισης νημάτων επιπέδου πυρήνα) αυξάνει το χρόνο εκτέλεσης. Η ύπαρξη του ελαχιστού αυτού φαινεται χαλάτρια χρησιμοποιώντας λογαριασμούς αριθμού οριζόντιο άξονα. Συμπερίλαμβανε ή μεγάλη χρονική διάρκεια της πράξης του παλλαπλασιασμού μεγάλων πινάκων (ιδεατερα σειριακά) που χρησιμοποιούμε να έχουμε εμφανή αποτελέσματα- δεν επέτρεπε για πρακτικάς λόγους μεγάλο αριθμό πειραματικών επαναλήψεων που στατιστικά θα εκτόπιζε την τυχαίοτητα ή την ακρίβεια των πειραματικών μετρήσεων.

Συνημμένα. Επισυνάπτεται ο πηγάιος χώδικας σε μορφή χειμένου. Επίσης σε διακέκτη επισυνάπτονται το παρόν κείμενο σε ηλεκτρονική μορφή, οι έξοδοι από τις εκτελέσεις των προγραμμάτων και οι αντίστοιχες μετρήσεις, το Makefile, χαθώς και όλοι οι πηγαίοι χώδικες και τα αντίστοιχα εκτελέσιμα (και η βιβλιοθήκη).