Πανεπιστήμιο Πατρών
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής
Τομέας Λογικού 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 psthreads (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 κόστος 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 πραγματικό ευκταίο παραλληλισμό και τα οποία ονομάζουμε ιδεατούς 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 δεδομέων και τις συναρτήσεις που όρισε το σύστημα διεπαφής με τον προγραμματιστή (Application Programming Interface) των προδιαγραφών, καθώς και τύπους δεδομέων και συναρτήσεις για εσωτερική χρήση. Αναλυτικά αυτά είναι:

Τύποι δεδομένων του API.

- psthread_t για νήμα επιπέδου χρήστη. Στον περιγραφέα αυτό συμπεριλάβαμε δείκτη στη στοίβα του νήματος, δείκτη στο προηγούμενο και στο επόμενο νήμα στην ουρά, χώρο για να αποθηκεύεται το περιβάλλον του νήματος, έναν ακέραιο 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 βαίνουν καλώς), δείκτη στη συνάρτηση που εκτελείται κατά την εκκίνηση του νήματος, δείκτη στα ορίσματα αυτής, δείκτη στο νήμα που έχει κάνει join στο νήμα αυτό (εάν έχει σύμβει κάτι τέτοιο) (εξυπηρετεί στην αλληλεπίδραση των psthread_exit(), psthread_join()) και lock που συγχρονίζει τις αλλαγές στα πεδία του περιγραφέα και που πάλι εξυπηρετεί την αλληλεπίδραση των psthread_exit(), psthread_join(), αφού εξασφαλίζει ότι η psthread_join() δε θα ελευθερώσει την περιοχή μνήμης που αντιστοιχεί στον περιγραφέα του νήματος πριν η psthread_exit() για το αντίστοιχο νήμα δεν ολοκληρώσει την εκτέλεσή της, κάτι που θα προκαλούσε προβλήματα.

- psthread_lock_t για μεταβλητή συγχρονισμού. Χρησιμοποιήθηκε το Test and Test and Set Lock, volatile για λόγους ορθότητας και padded για λόγους επίδοσης.

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

Συναρτήσεις του API.

- int psthread_init(int kernel_threads) για την αρχικοποίηση των εσωτερικών δομών της βιβλιοθήκης και για τη δημιουργία του ζητούμενου αριθμού νημάτων επιπέδου πυρήνα (ιδεατών επεξεργαστών). Τα νήματα επιπέδου πυρήνα δημιουργήθηκαν με τις κλήσεις που ορίζει το πρότυπο POSIX threads για λόγους μεταφερσιμότητας, αν και υλοποιήθηκε δοκιμαστικά και η δημιουργία τους με τη βοήθεια της κλήσης 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 υποτυπώδες νήμα επιπέδου χρήστη, αντίστοιχο του ήδη εκτελούμενου κώδικα, το οποίο χρησιμοποιείται μόνο για το πρώτο context-switch και δεν τοποθετείται σε κάποια ουρά. Ακόμα δημιουργούμε άεργο νήμα για τον ιδεατό επεξεργαστή 0 πάλι που τον απασχολεί όταν δεν υπάρχουν χρήσιμα νήματα επιπέδου χρήστη προς εκτέλεση.

- int psthread_create(psthread_t ** thread_desc, void (start_routine)(void *), void *arg, int queue) για τη δημιουργία ενός νήματος επιπέδου χρήστη

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

- int psthread_join(psthread_t *thread_desc) για την αναστολή της εκτέλεσης ενός νήματος επιπέδου χρήστη έως ότου τερματίσει κάποιο άλλο συγκεκριμένο, οπότε και απελευθερώνονται οι αντίστοιχες εσωτερικές δομές του τερματίσαντος.

- void psthread_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 άεργοι

επεξεργαστές να ξεκινήσουν την εκτέλεση νήματος που καλεί την psthread_yield() πριν η εκτέλεση αυτής ολοκληρωθεί θα πρέπει να ληφθεί ειδική πρόνοια εκκαθάρισης του περιβάλλοντος στην MD_LONGJMP() ή η _ps_schedule() να εκτελείται ως ξεχωριστό 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 αποφυγή αδιεξόδων (μιας και τα νήματα είναι μη προεκτοπιστικά) 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(psthread_barrier_t *barrier, int thread_count) για την αρχικοποίηση φράγματος.
- int psthread_barrier_wait(psthread_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 ιδεατών επεξεργαστών (νημάτων επιπέδου πυρήνα) που θα χρησιμοποιηθούν, όπως τον ορίζει 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 έτοιμα προς εκτέλεση. Χρησιμοποιήθηκε μία 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 στοίβα ενός νήματος επιπέδου χρήστη (παρεχόταν έτοιμη). Το μέγεθος κάθε στοίβας 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 άεργο 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_acttthr_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 διαπιστωθεί 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 ορίσματά της, αφού η MD_INIT_CONTEXT() δέχεται ως όρισμα συνάρτηση εκκίνησης χωρίς ορίσματα (οπότε και της παιρνάμε την _πς_ρυν(οιδ).

- int _ps_find_vpnum(void) για την εύρεση του ιδεατού επεξεργαστή στον οποίο εκτελούμαστε, στοιχείο απαραίτητο για την εύρεση του αντίστοιχου ενεργού ή άεργου νήματος.

- void _ps_idle(void) για την απασχόληση του επεξεργαστή όσο δεν υπάρχει χρήσιμη εργασία και η εφαρμογή δεν έχει τερματίσει.

- void _ps_schedule(void) για τη χρονοδρομολόγηση, τη μετάβαση του επεξεργαστή από ένα νήμα επιπέδου χρήστη σε άλλο.

- void *_ps_vp_init(void *vpnumber) για την αρχικοποίηση των ιδεατών επεξεργαστών. Περιλαμβάνει ξανά τη δημιουργία υποτυπώδους νήματος επιπέδου χρήστη, αντίστοιχο του ήδη εκτελούμενου κώδικα, το οποίο χρησιμοποιείται μόνο για το πρώτο context-switch και δεν τοποθετείται σε κάποια ουρά και τη δημιουργία άεργου 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 στοίβας.

- void _ps_lock_init(psthread_lock_t *thr_ttas_lock_ptr) για την αρχικοποίηση εσωτερικής μεταβλητής συγχρονισμού.

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

- void _ps_unlock(psthread_lock_t *thr_ttas_lock_ptr) για την απελευθέρωση εσωτερικής μεταβλητής συγχρονισμού.

- void _ps_barrier_init(psthread_barrier_t *thr_incr_barrier, int processors) για την αρχικοποίηση εσωτερικού φράγματος.

- void _ps_barrier(psthread_barrier_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, psthread_t *t) για την εισαγωγή στοιχείου στην κορυφή ουράς.
- void _ps_queue_append(ps_queue_t *q, psthread_t *t) για την εισαγωγή στοιχείου στο τέλος ουράς.
- psthread_t * _ps_queue_removefirst(ps_queue_t *q) για τη λήψη στοιχείου από την κορυφή ουράς.
- psthread_t * _ps_queue_remove(ps_queue_t *q, psthread_t *t) για την εξαγωγή συγκεκριμένου στοιχείου (ανεξάρτητα από τη θέση του) από ουρά.

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

Υλοποίηση πολυνηματικών προγραμμάτων.

Υλοποίηση βοηθητικών προγραμμάτων ελέγχου. Προκειμένου να ελέγχουμε τη σωστή λειτουργία της βιβλιοθήκης αναπτύσσαμε 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 υλοποιούσαμε. 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 αλλαγές που επέφεραν οι κλήσεις των συναρτήσεων της βιβλιοθήκης στην ουρά ήταν όντως οι επιθυμητές.

Το testlib καλεί τις συναρτήσεις του API της βιβλιοθήκης και βασικές εσωτερικές συναρτήσεις, όπως η _ps_find_vpnum(). Το πρόγραμμα αρχικοποιεί τη

βιβλιοθήκη, δημιουργεί νήματα επιπέδου χρήστη τα οποία αυτοπροσδιορίζονται, αυξάνουν ένα μετρητή και βγαίνουν και κατόπιν τα κάνει join. Ο έλεγχος του μετρητή και η εκτύπωση μηνυμάτων μας επιτρέπουν να δούμε αν όλα τα νήματα εκτελέστηκαν σωστά. Αρχικά δοκιμάσαμε τη βιβλιοθήκη σε έναν ιδεατό επεξεργαστή και στη συνέχεια σε περισσότερους. Προκειμένου και ο ιδεατός επεξεργαστή 0 να μπορεί να εκτελεί νήματα επιπέδου χρήστη, καλεί την pthread_yield() ή την pthread_join() μετά την αρχικοποίηση της βιβλιοθήκης και τη δημιουργία νημάτων επιπέδου χρήστη για να μπει στο βρόχο απόσπασης νημάτων από τη ready-queue.

Υλοποίηση εφαρμογής πολλαπλασιασμού πινάκων. Προκειμένου να ελέγξουμε την αποδοτικότητα και τη λειτουργικότητα της βιλιοθήκης υλοποιούμε εφαρμογή (matmul.c) πολλαπλασιασμού (υπολογισμού του εσωτερικού γινομένου) δύο πινάκων αριθμών κινητής υποδιαστολής διπλής ακρίβειας, μεγέθους 2048x2048, αρχικοποιημένους με τυχαίους αριθμούς. Το πρόγραμμα δέχεται ως είσοδο τον αριθμό των ιδεατών επεξεργαστών και των αριθμό νημάτων επιπέδου χρήστη και την εντολή να εκτελέσει ή όχι τον πολλαπλασιασμό και σειριακά και να επαληθεύσει το παράλληλο αποτέλεσμα (σύμπτωση σε 3 δεκαδικά ψηφία). Χρησιμοποιούμε τον ίδιο αλγόριθμο πολλαπλασιασμού πινάκων για τον παράλληλο και σειριακό πολλαπλασιασμό, ο οποίος χρησιμοποιώντας την τοπικότητα στη μνήμη επιτρέπει αυξημένη επίδοση. Στην περίπτωση του παράλληλου πολλαπλασιασμού κάθε νήμα επιπέδου χρήστη αναλαμβάνει να υπολογίσει ένα τμήμα του πίνακα-αποτελέσματος. Οι μετρήσεις χρόνου περιλαμβάνουν στην περίπτωση του παράλληλου πολλαπλασιασμού το χρόνο αρχικοποίησης της βιβλιοθήκης, το χρόνο εκτέλεσης των νημάτων και το χρόνο τερματισμού και απελευθέρωσης των δομών τους.

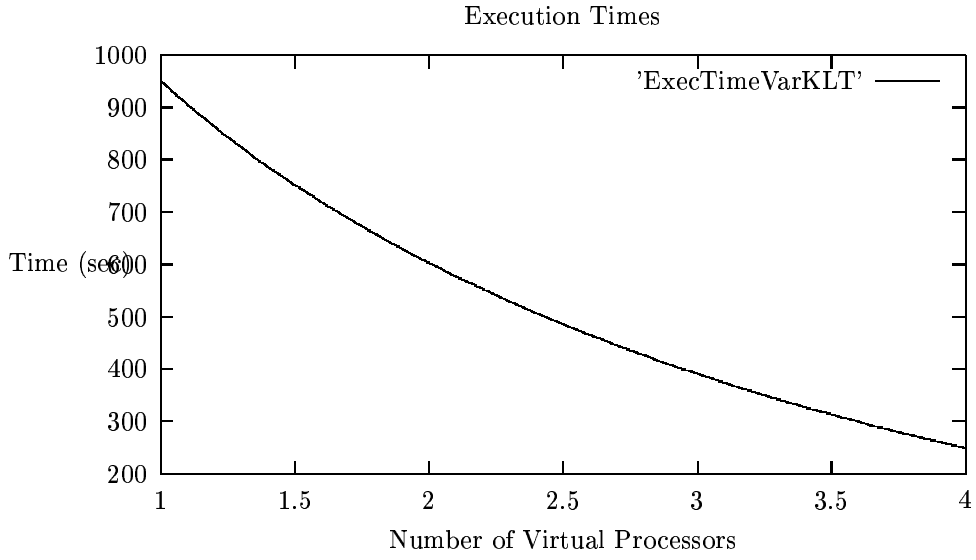
Makefile, ορίσματα γραμμής εντολών. Προκειμένου να απλοποιήσουμε τη διαδικασία δημιουργίας της βιβλιοθήκης και των εφαρμογών και να αποφεύγουμε άσκοπες μεταγλωττίσεις πηγαίου κώδικα που δεν έχει τροποποιηθεί χρησιμοποιήσαμε τη make και γράψαμε το κατάλληλο Makefile. Εκεί φαίνονται και οι παράμετροι που 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 συνδέτη (gcc): -L$(LIB_PATH) (για τον εντοπισμό της βιβλιοθήκης που υλοποιήσαμε) -lpthread (για σύνδεση με τη βιβλιοθήκη POSIX νημάτων) -lpsthreads (για σύνδεση με τη βιβλιοθήκη νημάτων που υλοποιήσαμε) για τη δημιουργία των εφαρμογών. Αντίστοιχα κατά τη μεταγλώττιση της βιβλιοθήκης περνάμε τις ακόλουθες παραμέτρους στο μεταγλωττιστή:

-mcpu=ultrasparc -O6 και προαιρετικά τη -Wall. Είναι φανερό ότι πρέπει να δηλώσουμε τον επεξεργαστή του οποίου την assembly χρησιμοποιουμε στις συναρτήσεις υλοποίησης των ατομικών πράξεων.

Μετρήσεις. Οι μετρήσεις -όπως και η υλοποίηση- πραγματοποιήθηκαν σε συμμετρικό πολυεπεξεργαστικό σύστημα κοινής μνήμης με 4 επεξεργαστές UltraSparcII @ 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 ο φόρτος του συστήματος -όπως δίνεται από την uptime- ήταν χαμηλός, ($\lesssim$0.5). Πειραματιστήκαμε με τον αριθμό των ιδεατών επεξεργαστών για να υπολογίσουμε τη χρονοβελτίωση που επιτυγχάνεται και με τον αριθμό των νημάτων επιπέδου χρήστη για να υπολογίσουμε τη βελτιστοποίηση του χρόνου εκτέλεσης. (Δίνοντας ps -L κατά τη διάρκεια του χρόνου εκτέλεσης είδαμε τα νήματα που δημιουργήσαμε και τα επιπλέον νήματα ελέγχου που κατασκευάζει αυτόματα η βιβλιοθήκη νημάτων χρόνου εκτέλεσης του λειτουργικού συστήματος και τα οποία δεν παίρνουν επεξεργαστικό χρόνο.) Ακολουθούν οι μετρήσεις υπό τη μορφή πινάκων και διαγραμμάτων και σχολιασμός - ερμηνεία των αποτελεσμάτων:

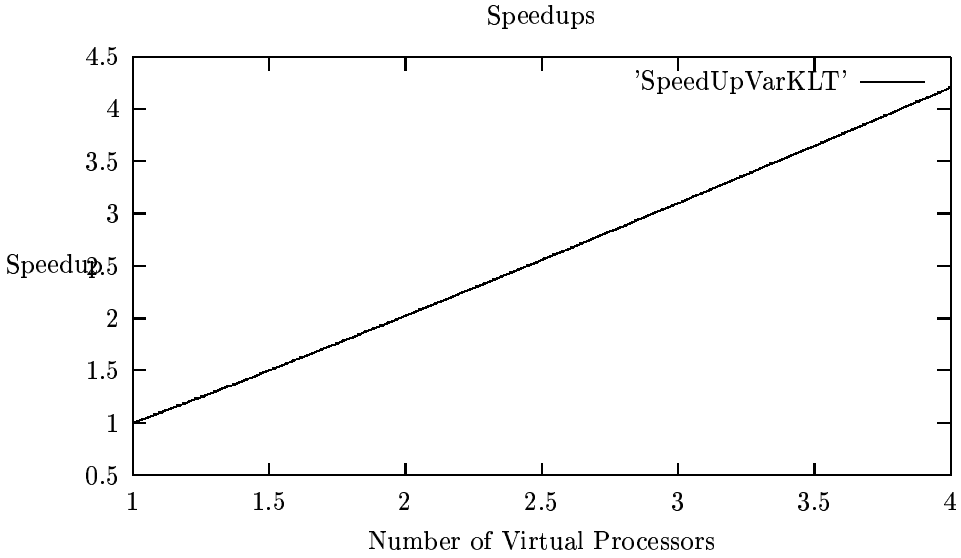
Χρόνος εκτέλεσης για μεταβλητό αριθμό εικονικών επεξεργαστών. Στην περίπτωση αυτή μεταβάλλαμε τον αριθμό των εικονικών επεξεργαστών από 1 έως τον αριθμό των φυσικών επεξεργαστών. Ορίσαμε τον αριθμό των νημάτων επιπέδου χρήστη ίσο με τον αριθμό των νημάτων επιπέδου πυρήνα (ιδεατών επεξεργαστών). Μετρήσαμε το χρόνο παράλληλης εκτέλεσης.

$NumOfVPs(= klts = ults)$	$ExecutionTime(sec)$
1	951.49
2	482.22
4	249.22



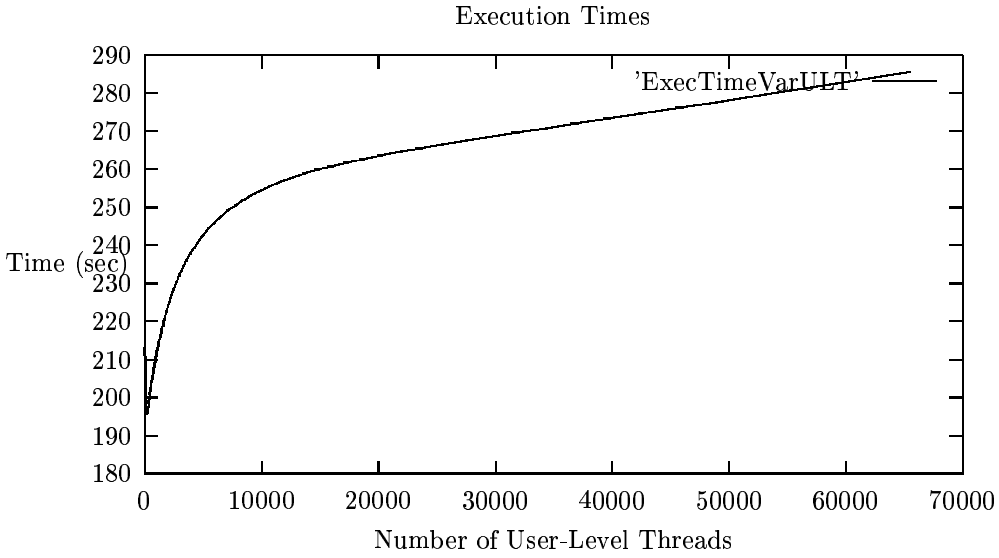
Χρονοβελτίωση για μεταβλητό αριθμό εικονικών επεξεργαστών. Ομοίως με την παραπάνω περίπτωση μεταβάλλαμε τον αριθμό των εικονικών επεξεργαστών και μετρώντας χρόνους παράλληλης και σειριακής εκτέλεσης υπολογίσαμε τη χρονοβελτίωση (speedup) για κάθε περίπτωση.

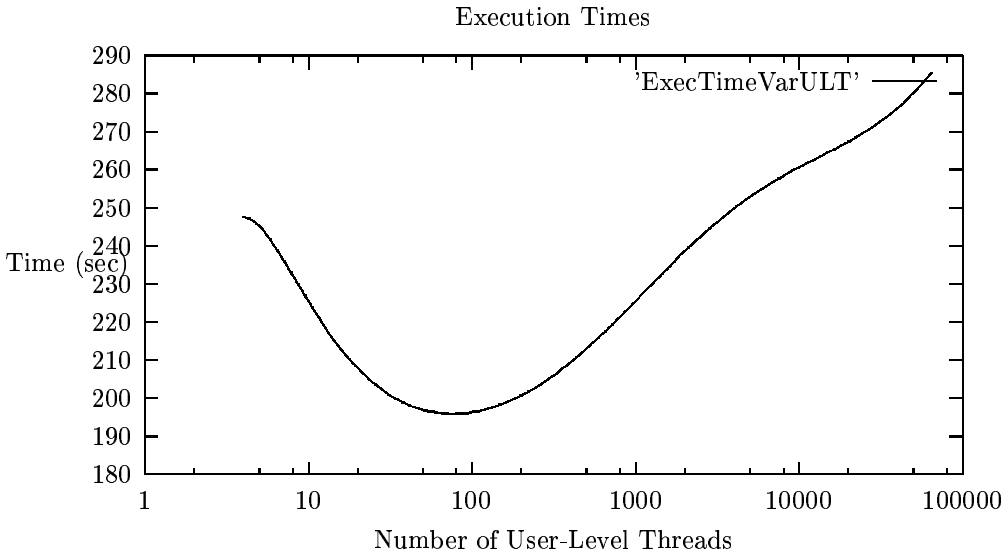
$NumOfVPs(= klts = ults)$	$SeqTime(sec)$	$ParTime(sec)$	$Speedup$
1	951.13	951.49	0.999
2	952.86	482.22	1.976
4	1047.78	249.22	4.204



Χρόνος εκτέλεσης για σταθερό αριθμό εικονικών επεξεργαστών (4) και μεταβλητό αριθμό νημάτων επιπέδου χρήστη. Στην περίπτωση αυτή μεταβάλλαμε τον αριθμό των νημάτων επιπέδου χρήστη, διατηρώντας σταθερό τον αριθμό των ιδεατών επεξεργαστών ίσο με τον αριθμό των φυσικών επεξεργαστών (4). Ξεκινήσαμε από τόσα νήματα όσα και ο βαθμός παραλληλισμού (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 χρήστη (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.

$NumOfUser-LevelThreads$	$ExecutionTime(sec)$
sequential	1047.78
4	247.60
8	247.60
16	190.77
32	191.64
64	189.98
128	190.74
256	194.08
512	203.23
1024	220.92
2048	253.86
4096	256.64
8192	258.49
16384	263.39
32768	269.48
65536	285.57


Execution Times
'ExecTimeVarULT'
Time (sec)
290
280
270
260
250
240
230
220
210
200
190
180
0
10000
20000
30000
40000
50000
60000
70000
Number of User-Level Threads


Execution Times
'ExecTimeVarULT'
Time (sec)
290
280
270
260
250
240
230
220
210
200
190
180
1
10
100
1000
10000
100000
Number of User-Level Threads

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 χρονοβελτιώσεις που επιτυγχάνονται. Ο χρόνος σειριακής εκτέλεσης παρατηρούμε ότι μεταβάλλεται -αν και ελάχιστα- στις εκτελέσεις του προγράμματος με αριθμό ιδεατών επεξεργαστών μεγαλύτερο του 1, πιθανώς μέσα στα πλαίσια της πειραματικής ακρίβειας. Έτσι και η υπεργραμμική χρονοβελτίωση που παρατηρούμε για 4 ιδεατούς επεξεργαστές πιθανά να είναι τυχαία, αν και δεν αποκλείεται να οφείλεται σε εκμετάλλευση της τοπικότητας της μνήμης ή σε βελτιστοποιήσεις του μεταγλωττιστή. Σχετικά με το μεταβλητό αριθμό νημάτων επιπέδου χρήστη στην περίπτωση που χρησιμοποιούμε ιδεατούς επεξεργαστές όσους και φυσικούς, μπορούμε να παρατηρήσουμε ότι μέχρι έναν αριθμό νημάτων επιπέδου χρήστη (64) έχουμε μείωση του χρόνου εκτέλεσης, μιας και εκμεταλλευόμαστε το λεπτά καταμερισμένο παραλληλισμό της εφαρμογής. Κατόπιν το κόστος διαχείρισης του μεγάλου αριθμού νημάτων επιπέδου χρήστη (αν και μικρότερο από αυτό της διαχείρισης νημάτων επιπέδου πυρήνα) αυξάνει το χρόνο εκτέλεσης. Η ύπαρξη του ελαχίστου αυτού φαίνεται καλύτερα χρησιμοποιώντας λογαριθμικό οριζόντιο άξονα. Σε κάθε περίπτωση σημειώνουμε τη μεγάλη χρονική διάρκεια της πράξης του πολλαπλασιασμού μεγάλων πινάκων (ιδιαίτερα σειριακά) που -αν και απαραίτητη προκειμένου να έχουμε εμφανή αποτελέσματα- δεν επέτρεπε για πρακτικούς λόγους μεγάλο αριθμό πειραματικών επαναλήψεων που στατιστικά θα εκτόπιζε την τυχαιότητα ή την ανακρίβεια των πειραματικών μετρήσεων.

Συνημμένα. Επισυνάπτεται ο πηγαίος κώδικας σε μορφή κειμένου. Επίσης σε δισκέττα επισυνάπτονται το παρόν κείμενο σε ηλεκτρονική μορφή, οι έξοδοι από τις εκτελέσεις των προγραμμάτων και οι αντίστοιχες μετρήσεις, το Makefile, καθώς και όλοι οι πηγαίοι κώδικες και τα αντίστοιχα εκτελέσιμα (και η βιβλιοθήκη).