Πανεπιστήμιο Πατρών
Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Λογισμικό και Προγραμματισμός σε Συστήματα Υψηλών Επιδόσεων

ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ 1

2011-2012

Βασιλάκης Φίλιππος
3895
vasilakis@ceid.upatras.gr

Κλησιάρης Δημήτρης
3943
klisiaris@ceid.upatras.gr

Εισαγωγή

Στην άσκηση αυτή καλούμαστε να υλοποιήσουμε μια συνάρτηση υψηλού επιπέδου για την αυτόματη παραλληλοποίηση βρόγχων με νήματα διαμοιράζοντας τον φόρτο έργου είτε σε λεπτά (fine-grain) είτε αδρά (coarse-grain) ανάλογα με το ποσοστό παραλληλοποίησης που επιθυμούμε.

Το compile γίνεται με την εντολή make all στον κατάλογο των αρχείων πηγαίου κώδικα και η εκτέλεση του με την ./bench

Η συνάρτηση που υλοποιήθηκε είναι η εξής:

int pthread_execute_loop (void (*func)(void *arg, int low, int high), void *arg, int policy, int chunk, int nthreads, int low, int high)

Τα ορίσματα είναι τα εξής:

- void (*func)(void *arg, int low, int high): Η συνάρτηση με τα 3 ορίσματά της. Συγκεκριμένα τα ορίσματά της είναι το *arg το οποίο είναι όρισμα που χρησιμοποιείται μόνο από τον προγραμματιστή ενώ τα low και high ορίζουν την αρχή και το τέλος των επαναλήψεων του βρόγχου που υλοποιείται μέσα στην συνάρτηση
- int policy: Η παράμετρος αυτή χρησιμοποιείται για τον τύπο δρομολόγησης που θα χρησιμοποιηθεί από τα νήματα.Συγκεκριμένα επιτρέπονται μόνο οι τιμές
  - DYNAMIC
  - STATIC
- int chunk: Το chunk ορίζεται ως το φόρτο εργασίας, δηλαδή επαναλήψεις, που θα πάρει το κάθε νήμα και χρησιμοποιείται μόνο στο dynamic τύπο δρομολόγησης.
- int nthreads: Ο αριθμός των threads που θα τρέξουν τον βρόγχο
- int low, int high: Η αρχή και το τέλος του δείκτη του βρόγχου

Η υλοποίησή μας είναι στα αρχεία loop.h και loop.c. Ουσιαστικά ένας χρήστης (προγραμματιστής) έχοντας τα 2 αυτά αρχεία μπορεί να τα κάνει #include στον κώδικα του και να χρησιμοποιήσει την συνάρτηση υψηλού επιπέδου.

Τα αρχεία που έχουν αποσταλεί εκτός από τα loop.h και loop.c που θα εξηγηθεί η λειτουργία τους παρακάτω είναι και τα εξής:

- **Makefile**: Το makefile μας για εξοικονόμηση χρόνου κατα το compile

- **bench_functions.h/bench_functions.h**: Μέσα σε αυτό το αρχείο υπάρχουν υλοποιήσεις συναρτήσεων που δοκιμάσαμε διεξοδικά για την υλοποίηση της συνάρτησης που μας ζητείται. Επίσης περιέχονται κάποιες δευτερεύουσες συναρτήσεις που θα μπορούσε να είχε υλοποιήσει και ο χρήστης όπως συναρτήσεις για την μέτρηση του χρόνου και την εξαγωγή πληροφοριών στην οθόνη.

- **main.c**: Ουσιαστικά αυτό το αρχείο υλοποιεί την main() συνάρτηση και λαμβάνει τα εξής ορίσματα:

    1. *policy*: Μπορεί να είναι είτε 1 για STATIC είτε 2 για DYNAMIC.

        - STATIC Η δρομολόγηση αυτή χωρίζει τις επαναλήψεις(N) σε P ομάδες όπου P είναι ο αριθμός των νημάτων. Δηλαδή το κάθε νήμα λαμβάνει N/P επαναλήψεις. Εάν απομένουν επαναλήψεις αυτές κατανέμονται από μια στα πρώτα νήματα ώστε να έχουμε όσο το δυνατό καλύτερη κατανομή του φόρτου εργασίας

        - DYNAMIC: Κατα την επιλογή αυτής της δρομολόγησης κάθε φορά που ένα νήμα βρίσκεται σε απραξία δεσμεύει και εκτελεί τις chunk επόμενες επαναλήψεις. Αυτό ουσιαστικά σημαίνει οτι κάθε νήμα συνεχίζει στις επόμενες chunk επαναλήψεις μέχρι να τελειώσει ο συνολικός φόρτος εργασίας.

2. *chunk*: Ορίζει το φόρτο εργασίας που θα έχει το κάθε νήμα. Χρησιμοποιείται μόνο στην DYNAMIC δρομολόγηση. Στην STATIC δεν χρησιμοποιείται. Αν οριστεί -1 τότε ορίζεται αυτόματα στο 1

3. *num_of_threads*: Ορίζει τον αριθμό των νημάτων που θα εκτελέσουν το παράλληλο βρόγχο.

Να σημειωθεί οτι στις συναρτήσεις ploop_func και vector_loop_func του αρχείου bench_functions.c, έχει χρησιμοποιηθεί ένα επιπλέον “τεχνητό” κόστος το οποίο κοστίζει περίπου 20 millisecond ανα επανάληψη. Αυτό έγινει προκειμένου να επιδείξουμε την αποδοτικότητα της υλοποίησης της συνάρτησής μας αφού μόνη της η πρόσθεση των διανυσμάτων εκτελείται ταχύτατα στα σύγχρονα υπολογιστικά συστήματα(΄οπου έγιναν και οι μετρήσεις).

Οι μετρήσεις έγιναν στο υπολογιστικό σύστημα **etocles** του **hpclab** , το οποίο διαθέτει **4 επεξεργαστές Intel Xeon 5130 στα 2Ghz** και στο οποίο έτυχε να έχουμε πρόσβαση. Προτιμήθηκε το συγκεκριμένο πολύ-επεξεργαστικό σύστημα διότι δεν είχαμε στη διάθεση μας κάποιον προσωπικό υπολογιστή με πολυπύρηνο επεξεργαστή και το συγκεκριμένο σύστημα είχε χρησιμοποιηθεί και παλαιότερα για μετρήσεις στο μάθημα της παράλληλης επεξεργασίας. Η ανάπτυξη του κώδικα έγινε σε συστήματα με intel c2d e7400, 4GBram,Ubuntu 11.10 και intel Pentium m 1,73GHz, 1GBram, Lubuntu 11.10.

Βοηθητικά αρχεία

Αν και ουσιαστικά η ζητούμενη υλοποίηση σχετίζονταν με την συνάρτηση *pthread_execute_loop*, για τις δοκιμές που έπρεπε να γίνουν (έλεγχος ορθότητας αποτελεσμάτων, μετρήσεις αποδοτικότητας κλπ) ήταν απαραίτητο να υλοποιηθεί ένα σύνολο βοηθητικών συναρτήσεων.

Οι συναρτήσεις αυτές περιέχονται στα αρχεία **bench_functions.c, bench_functions.h** και ουσιαστικά είναι συναρτήσεις με εκτέλεση βρόγχων που ένας προγραμματιστής έχει υλοποιήσει και επιθυμεί να τις παραλληλοποιήσει. Ωστόσο εμείς πέρα από αυτές έχουμε υλοποιήσει και κάποιες που βοηθούν στη διαδικασία των μετρήσεων.

Αναλυτικά:

- `void loop_func(void *arg, int start, int stop);`
  Πολύ απλή συνάρτηση που υλοποιεί έναν βρόγχο και εμφανίζει σε κάθε επανάληψη τον αριθμό της επανάληψης και έναν αριθμό που έχει περαστεί από το όρισμα αυξημένο κατά τον αριθμό της επανάληψης. Η συνάρτηση χρησιμοποιήθηκε για μικρό αριθμό επαναλήψεων για να διαπιστώσουμε την σωστή εκτέλεση των επαναλήψεων από τα νήματα.

- `void ploop_func(void *arg, int low, int high);`
  Η δοσμένη από την εκφώνηση συνάρτηση που γράφει σε έναν πίνακα (ή τον αρχικοποιεί με) την θέση του στοιχείου αυξημένο κατά 10. Επειδή λόγω της απλότητας της συνάρτησης θα έπρεπε να εκτελεστούν εκατομμύρια επαναλήψεις προκειμένου να πάρουμε χρόνο εκτέλεσης της τάξης του δευτερολέπτου που σημαίνει τεράστιος πίνακας, για να διευκολύνουμε τις μετρήσεις προσθέσαμε ένα επιπλέον τεχνητό κόστος περίπου 20millisec σε κάθε επανάληψη.

- `void vector_loop_func(void * arg, int low, int high);`
  Η συνάρτηση αυτή υλοποιεί την πράξη C=A+B , όπου A,B,C πίνακες. Στην υλοποίηση μας απαιτείται πιο πριν να έχει κατασκευαστεί μια δομή με δείκτες προς τους τρεις πίνακες και στην συνάρτηση περνάμε έναν δείκτη προς τη δομή αυτή. Και εδώ σε κάθε επανάληψη προσθέσαμε τεχνητό κόστος. Επειδή θεωρούμε ότι μία τέτοια πρόσθεση θα έχει νόημα μόνο αν οι A και B έχουν τιμές έπρεπε οι A και B να αρχικοποιηθούν πρώτα. Η παρακάτω συνάρτηση υλοποιήθηκε γι αυτό το σκοπό.

- `void init_vectors(void * arg, int size, int range);`
  Αναλαμβάνει να δεσμεύσει μνήμη και αρχικοποιήσει τους πίνακες που θα χρησιμοποιηθούν για την πρόσθεση. Χρησιμοποιούνται τυχαίες τιμές στο εύρος 1 έως range για την αρχικοποίηση.

- `double return_time(void);`
  Επιστρέφει την ώρα εκείνη τη στιγμή με χρήση της gettimeofdate()

- `void one_milli(void);`
  Υλοποιεί έναν βρόγχο 28000 επαναλήψεων που εκτελεί τις πράξεις sqrt και pow και θεωρητικά δίνει καθυστέρηση 1 millisec στο σύστημα που έγιναν οι μετρήσεις.
- `void comp(int millis);`
  Καλεί τη συνάρτηση one_milli όσες φορές καθορίζεται από το όρισμά του.

Στο αρχείο **main.c** περιέχεται η συνάρτηση main() που ανάλογα με την δοκιμή που θέλουμε να κάνουμε γράφουμε την συνάρτηση που θέλουμε να εκτελεστεί. Στο αρχείο που παραδίδουμε καλούνται οι ploop_func και vector_loop_func μέσω της pthread_execute_loop και μετριούνται οι χρόνοι εκτέλεσής τους. Είναι δυνατό να χρησιμοποιηθούν τα arguments της main για αλλαγή ορισμένων παραμέτρων της loop όπως ειπώθηκε παραπάνω αλλά όχι για τα κάτω και πάνω όρια εκτέλεσης των επαναλήψεων. Για να γίνει αυτό θα πρέπει να γίνουν οι αλλαγές κατευθείαν στον κώδικα της main και compile.

Τέλος το **Makefile** χρησιμοποιείται για την γρήγορη μετάφραση του πηγαίου κώδικα και υποστηρίζει τις εντολές:

make all : κανει compile τον κώδικά μας και δημιουργεί το εκτελέσιμο αρχείο **bench**

make clean : διαγράφει τα ενδιάμεσα object αρχεία που δημιουργούνται κατά την μετάφραση

Εναλλακτικά μπορεί να γίνει compile με την εντολή `gcc main.c loop.c bench_functions.c -o bench -W -Wall -Wextra -lpthread –lm`

Η συνάρτηση pthread_execute_loop

Η κύρια συνάρτηση της άσκησης υλοποιείται στα αρχεία loop.c και loop.h. Η λειτουργία της περιγράφεται αποκλειστικά σε αυτά τα δύο αρχεία και δεν έχει εξαρτήσεις με κανένα άλλο αρχείο στον κώδικα. Αυτό σημαίνει ότι ο προγραμματιστής χρειάζεται απλά να κάνει include το loop.h και να συμπεριλάβει το loop.c στην μετάφραση ώστε να χρησιμοποιήσει την συνάρτηση.

Έχουν ήδη ειπωθεί κάποια πράγματα για τα ορίσματα της συνάρτησης στην εισαγωγή. Εδώ θα αναφερθούμε στον τρόπο λειτουργίας της εσωτερικά και τις συναρτήσεις που έχουμε υλοποιήσει για χρήση από την ίδια την συνάρτηση.

Χρησιμοποιήθηκαν δύο διαφορετικές δομές που κρατάνε την πληροφορία που χρειάζεται το κάθε νήμα ανάλογα με την πολιτική.

Στην *στατική πολιτική* η δομή **pthread_data** κρατάει δείκτες προς την συνάρτηση και το όρισμα της συνάρτησης και τους ακεραίους low και high που προσδιορίζουν ποιες επαναλήψεις έχουν ανατεθεί στο συγκεκριμένο νήμα.

Στην *δυναμικη πολιτική* η δομή **dynamic_pthread_data** κρατάει δείκτες προς την συνάρτηση και το όρισμα της συνάρτησης και τους ακεραίους chunk και global_high που προσδιορίζουν πόσες επόμενες επαναλήψεις θα δεσμεύσει το συγκεκριμένο νήμα και ποιο είναι το συνολικό άνω όριο των επαναλήψεων. Τέλος κρατάει έναν δείκτη στην μεταβλητή chunk_counter που δείχνει μέχρι ποια επανάληψη έχει δεσμευτεί μέχρι τώρα. Έτσι όταν ένα νήμα βρεθεί σε απραξία κοιτάει τον chunk_counter και δεσμεύει τις chunk επόμενες επαναλήψεις αλλάζοντας και την τιμή του μετρητή στην νέα επανάληψη. Για την πρόσβαση των νημάτων στην τιμή του μετρητή έχει χρησιμοποιηθεί αμοιβαίος αποκλεισμός.

Οι συναρτήσεις:
`set_up_pthread_data()` και `set_up_dynamic_pthread_data()` χρησιμοποιούνται από την loop αρχικά ανάλογα με την πολιτική και αναλαμβάνουν με βάση τα ορίσματα που έχουν δοθεί από τον προγραμματιστή να δεσμεύσουν μνήμη και να αρχικοποιήσουν τις δομές με τα κατάλληλα δεδομένα για κάθε νήμα. Στη στατική πολιτική η set_up_pthread_data αναθέτει σε κάθε νήμα ακριβώς το σύνολο επαναλήψεων που θα εκτελέσει. Αυτό γίνεται χωρίζοντας τις επαναλήψεις σε ομάδες ίσες με τα νήματα με αριθμό επαναλήψεων N/P. Αν ο αριθμός αυτός είναι ακέραιος όλα τα νήματα εκτελούν τον ίδιο αριθμό επαναλήψεων. Αν όχι τότε σε κάθε περίπτωση ο αριθμός των επιπλέον επαναλήψεων θα είναι μικρότερος από τον αριθμό των νημάτων. Τότε αναθέτουμε στα πρώτα νήματα από μία επιπλέον επανάληψη έως ότου δεν υπάρχουν πλέον επιπλέον επαναλήψεις. Έτσι πετυχαίνουμε έναν καλό καταμερισμό της εργασίας.

Στη συνέχεια μέσω της pthread_create δημιουργούνται τα νήματα που θα χρησιμοποιηθούν και τρέχουν πάλι διαφορετική συνάρτηση ανάλογα με την πολιτική με όρισμα την δομή του κάθε νήματος.

Οι συναρτήσεις που τρέχουν τα νήματα ανάλογα με την πολιτική είναι οι εξής:

- `stat_pfunc()`
  Καλείται από τα νήματα σε στατική πολιτική. Απλή υλοποίηση αφού η πληροφορία που χρειάζονται τα νήματα υπάρχει στις δομές τους. Η συνάρτηση απλά τρέχει την συνάρτηση που πρέπει να τρέξει το νήμα με κάτω και πάνω όρια που έχουν υπολογιστεί και αποθηκευτεί στα structs.

- `dyn_pfunc()`
  Πιο περίπλοκη υλοποίηση αφού εδώ γίνεται δυναμικά έλεγχος και ανάθεση ορίων από το κάθε νήμα τη στιγμή της εκτέλεσης. Χρησιμοποιήθηκε μια while που εκτελείται μέχρι να τελειώσουν οι επαναλήψεις και ένας μετρητής chunk_counter που δείχνει μέχρι πια επανάληψη έχει δεσμευτεί. Με έλεγχο αμοιβαίου αποκλεισμού κάθε νήμα ελέγχει μέσω του μετρητή μέχρι ποια επανάληψη έχει δεσμευτεί και αναλαμβάνει να εκτελέσει τις επαναλήψεις από εκείνο το σημείο έως και τις chunk επόμενες. Αλλάζει την τιμή του μετρητή στη νέα τιμή ώστε να δείχνει μέχρι ποια επανάληψη έχει δεσμευτεί τώρα και ξεκλειδώνει το mutex. Στη συνέχεια εκτελεί την αρχική συνάρτηση με τα όρια που έχει υπολογίσει. Να σημειωθεί ότι γίνονται και έλεγχοι για την περίπτωση που ο μετρητής ξεπεράσει τις μέγιστες επαναλήψεις, όπου το νήμα τότε τερματίζει.

Οι παραπάνω συναρτήσεις υλοποιούν ουσιαστικά την pthread_execute_loop.

Μετρήσεις

Θα μετρήσουμε την απόδοση της συνάρτησής μας μετρώντας χρόνους εκτέλεσης και υπολογίζοντας τις επιταχύνσεις (speedup).

Θα χρησιμοποιήσουμε και τις δύο bench συναρτήσεις που έχουμε φτιάξει με 1500 επαναλήψεις για τις δύο πολιτικές και 2 διαφορετικά chunk για την δυναμική πολιτική.
Οι μετρήσεις θα γίνουν για 1,2,4,8,16 νήματα
Θα ξανά αναφέρουμε εδώ ότι χρησιμοποιήθηκε τεχνητό κόστος σε κάθε επανάληψη ώστε να «βαρύνει» ο υπολογισμός που κάνουμε και να φανούν καλύτερα οι διαφορές.

Οι χρόνοι παρουσιάζονται σε seconds.

Οι ακολουθιακοί χρόνοι εκτέλεσης αν τρέξουμε κατευθείαν τις συναρτήσεις είναι:
Tsec(ploop)= 12,655
Tsec(vector)= 12,655

Και παρακάτω οι παράλληλοι χρόνοι μέσω της pthread_execute_loop

Νηματα	p_loop Stat	p_loop Dyn 20	p_loop Dyn 250	Vector Stat	Vector Dyn 20	Vector Dyn 250
1	12,654	12,737	12,738	12,653	12,737	12,737
2	6,332	6,375	6,392	6,331	6,376	6,391
3	4,229	4,253	4,274	4,223	4,255	4,275
4	3,171	3,193	4,236	3,171	3,189	4,237
8	3,176	3,197	3,22	3,293	3,266	3,223
16	3,209	3,782	5,87	3,408	3,456	6,412