Η πολυνηματική γλώσσα προγραμματισμού Cilk

Β. Καρακάσης

Ερευνητικά Θέματα Υλοποίησης Γλωσσών Προγραμματισμού
Μεταπτυχιακό Μάθημα (688), ΣΗΜΜΥ

Νοέμβριος 2009

Βασικά χαρακτηριστικά

- Γλώσσα παράλληλου προγραμματισμού βασισμένη στην C (προσθήκη κάποιων επιπλέον λέξεων-κλειδιών).
- Η απομάκρυνση των λέξεων-κλειδιών της Cilk οδηγεί σε *έγκυρο* σειριακό πρόγραμμα C, που ονομάζεται η *C έκθλιψη* (C elision) του προγράμματος Cilk.
- Υπεύθυνο για την πραγματική παραλληλοποίηση είναι το περιβάλλον εκτέλεσης (runtime environment) της Cilk.

Βασικά χαρακτηριστικά (συνέχεια)

- Παραλληλισμός βασισμένος σε εργασίες (task-level parallelism) με χρήση ουρών εργασιών και κατάλληλου χρονοδρομολογητή.
- Κατάλληλη για δυναμικό και ασύγχρονο παραλληλισμό.
- Κατάλληλη για μηχανές μοιραζόμενης μνήμης (shared memory) που παρέχουν συνέπεια μνήμης (memory consistency).
- Ισχυρό αλγοριθμικό υπόβαθρο για τον χρονοδρομολογητή.
  - Η επίδοση των προγραμμάτων Cilk μπορεί να προβλέπεται και θεωρητικά.

Παράδειγμα προγράμματος Cilk

Αριθμοί Fibonacci

```
#include <stdlib.h>
#include <stdio.h>
#include <cilk.h>

cilk int fib (int n)
{
    if (n < 2) return n;
    else {
        int x, y;
        x = spawn fib(n-1);
        y = spawn fib(n-2);
        sync;
        return (x+y);
    }
}
```

Παράδειγμα προγράμματος Cilk

Αριθμοί Fibonacci (συνέχεια)

```
cilk int main (int argc, char *argv[])
{
    int n, result;
    n = atoi(argv[1]);
    result = spawn fib(n);
    sync;
    printf("Result: %d\n", result);
    return 0;
}
```

Το μοντέλο προγραμματισμού της Cilk

Δύο βασικές λέξεις-κλειδιά:

spawn Δημιουργία μίας νέας ροής εκτέλεσης (υπάρχει μία σχέση γονέα-παιδιού μεταξύ καλούντος και καλουμένου).

sync Αναμονή για την ολοκλήρωση παιδιών (λειτουργίες σε μεταβλητές που επηρεάζονται άμεσα ή έμμεσα από την εκτέλεση των παιδιών δεν είναι ασφαλείς πριν από την χρήση της sync).

Το μοντέλο προγραμματισμού της Cilk

Ροές εκτέλεσης και ατομικότητα

- Μία ροή εκτέλεσης (thread) στην Cilk είναι ένα *μέγιστο* σύνολο εντολών που τελειώνουν με spawn, sync ή return.
- Οι ροές εκτέλεσης μίας διαδικασίας (procedure) εκτελούνται ατομικά μεταξύ τους.

Δεν υπάρχει ανάγκη από ρητά κλειδώματα (locks).

Παράδειγμα ανανέωσης χωρίς κλείδωμα

```
cilk int fib (int n)
{
    int x = 0;
    inlet void summer (int result)
    {
        x += result;
        return;
    }

    if (n < 2) return n;
    else {
        summer(spawn fib(n-1));
        summer(spawn fib(n-2));
        sync;
        return (x);
    }
}
```

Τα inlets επιτρέπουν πιο σύνθετο χειρισμό της επιστρεφόμενης τιμής κάποιας διαδικασίας που έχει γίνει spawn, και επιπλέον εκτελούνται ατομικά.

Χρονοδρομολόγηση εργασιών

Θεωρητικό υπόβαθρο

Έστω T_P ο χρόνος εκτέλεσης ενός προγράμματος Cilk σε P επεξεργαστές, τότε

- T_1 είναι ο χρόνος του συνολικού έργου (work), και
- T_∞ είναι το μέγεθος (σε μονάδες χρόνου) του κρίσιμου μονοπατιού (critical path).

Επομένως, ο χρόνος εκτέλεσης ενός προγράμματος Cilk σε P επεξεργαστές είναι

$$T_P = T_1/P + O(T_\infty)$$

$$T_P \leq T_1/P + c_\infty T_\infty,$$

όπου c_∞ είναι η *επιβάρυνση του κρίσιμου μονοπατιού* (critical path overhead).

Χρονοδρομολόγηση εργασιών

Θεωρητικό υπόβαθρο (συνέχεια)

- Η μέγιστη επιτάχυνση της επίδοσης (speedup) είναι

$$\overline{P} = T_1/T_\infty.$$

- Το *περιθώριο παραλληλισμού* (parallel slackness) ορίζεται ως ο λόγος $\overline{P}/P$.

Εάν $\overline{P}/P \gg c_\infty$ (δηλ. ο αριθμός των διαθέσιμων επεξεργαστών είναι σαφώς μικρότερος του διαθέσιμου παραλληλισμού), τότε

$$T_P \approx T_1/P.$$

Χρονοδρομολόγηση εργασιών

Θεωρητικό υπόβαθρο (συνέχεια)

- Εάν T_S είναι ο χρόνος εκτέλεσης της C έκθλιψης, τότε ισχύει

$$c_1 = T_1/T_S,$$

όπου c_1 είναι η *επιβάρυνση έργου* (work overhead) του προγράμματος Cilk.

- Ο χρόνος εκτέλεσης, επομένως, ενός προγράμματος Cilk σε P επεξεργαστές, όταν υπάρχει μεγάλο περιθώριο παραλληλισμού, είναι

$$T_P \approx c_1 T_S/P$$

Χρονοδρομολόγηση εργασιών

Θεωρητικό υπόβαθρο (συνέχεια)

- Εάν T_S είναι ο χρόνος εκτέλεσης της C έκθλιψης, τότε ισχύει

$$c_1 = T_1/T_S,$$

όπου c_1 είναι η *επιβάρυνση έργου* (work overhead) του προγράμματος Cilk.

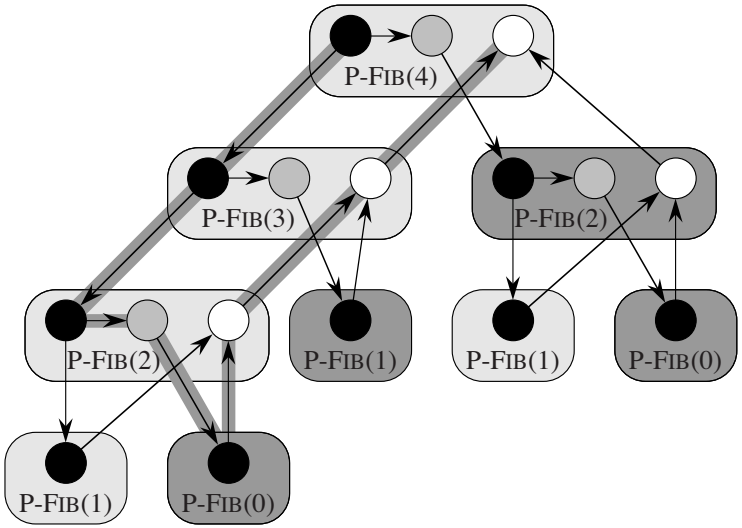
- Ο χρόνος εκτέλεσης, επομένως, ενός προγράμματος Cilk σε P επεξεργαστές, όταν υπάρχει μεγάλο περιθώριο παραλληλισμού, είναι

$$T_P \approx c_1 T_S/P$$

Βασική αρχή χρονοδρομολογητή

Ελαχιστοποίηση της επιβάρυνσης έργου.

Παράδειγμα θεωρητικού υπολογισμού T_1 και T_∞



- Κάθε spawn ή κοινή κλήση C δημιουργεί μία ακμή προς τα κάτω.
- Κάθε spawn δημιουργεί και μία οριζόντια ακμή συνέχισης (continuation edge).
- Κάθε sync δημιουργεί μία ακμή επιστροφής (return edge) με φορά προς τα πάνω.

- $T_1 = 17$ (πλήθος κόμβων).
- $T_\infty = 8$ (σκιασμένο μονοπάτι)
- Μέγιστη δυνατή επιτάχυνση: $17/8 = 2.125$

Η συνάρτηση P-Fib *είναι όμοια με την* `fib` *με την διαφορά ότι κάνει μόνο ένα spawn στη πρώτη αναδρομική κλήση.*

Θέματα υλοποίησης

Ουρές εργασιών και σχήμα κλοπής

- Κάθε φυσική μονάδα επεξεργασίας είναι ένας *εργάτης* (worker) που διατηρεί μία ουρά δύο άκρων με τις έτοιμες προς εκτέλεση διαδικασίες (double-ended ready queue – ready deque).
- Κάθε spawn σώζει το πλαίσιο ενεργοποίησης (activation frame) της γονικής διαδικασίας, το προωθεί στο τέλος της ουράς εργασιών του τρέχοντος εργάτη και συνεχίζεται η εκτέλεση.
- Μόλις αδειάσει η ουρά ενός εργάτη, τότε γίνεται *κλέφτης* (thief) και προσπαθεί να κλέψει την εργασία ενός άλλου εργάτη, που γίνεται πλέον *θύμα* (victim).

Θέματα υλοποίησης

Δημιουργία εργασιών

- Δύο κλώνοι για κάθε `cilk` διαδικασία: ένας *γρήγορος* και ένας *αργός*.
- Ο γρήγορος κλώνος είναι ουσιαστικά η C έκθλιψη της διαδικασίας με κάποιες δυνατότητες παραλληλισμού (προώθηση πλαισίων ενεργοποίησης στην ουρά εργασιών).
- Ο αργός κλώνος παρέχει πλήρη υποστήριξη παραλληλισμού (συγχρονισμός).

Θέματα υλοποίησης

Δημιουργία εργασιών (συνέχεια)

- Κάθε spawn εκτελεί τον γρήγορο κλώνο της διαδικασίας, και μόνο εάν η διαδικασία έχει κλαπεί, τότε γίνεται «μεταφορά» στον αργό κλώνο.
- Κάθε spawn αφότου σώσει το πλαίσιο ενεργοποίησης του πατέρα και το προωθήσει στην ουρά (push), καλεί συμβατικά (κλήση C) την διαδικασία που έγινε spawn στον κώδικα Cilk, και αμέσως μετά προσπαθεί να ανακτήσει (pop) το πλαίσιο ενεργοποίησής της. Εάν αποτύχει, σημαίνει ότι κλάπηκε.

Πλεονέκτημα

Το sync στον γρήγορο κλώνο δεν έχει καμία επιβάρυνση.

Θέματα υλοποίησης

Δημιουργία εργασιών (συνέχεια)

Παράδειγμα υλοποίησης του γρήγορου κλώνου

```
int fib (int n)
{
    fib_frame *f;
    f = alloc(sizeof(*f));
    f->sig = fig_sig;
    if (n < 2) {
        free(f, sizeof(*f));
        return n;
    } else {
        /* continuing... */
```

Θέματα υλοποίησης

Δημιουργία εργασιών (συνέχεια)

```
        /* cont'd */
        int x, y;
        f->entry = 1;
        f->n = n;
        *T = f;
        push();
        x = fib(n-1);
        if (pop(x) == FAILURE)
            return 0;
        /* ... */
        ;   /* sync is free */
        free(f, sizeof(*f));
        return (x+y);
    }
}
```

Θέματα υλοποίησης

Αλγόριθμος κλοπής εργασιών

Βασική αρχή

Πρώτα η δουλειά (work-first scheduling).

- Ελαχιστοποίηση της επιβάρυνσης έργου (ακόμα και) σε βάρος της επιβάρυνσης του κρίσιμου μονοπατιού.

Επιβάρυνση έργου Η επιπλέον δουλειά που καλείται να κάνει κάποιος εργάτης ως μέρος του πρωτοκόλλου χρονοδρομολόγησης (spawn, sync, abort).

Επιβάρυνση κρίσιμου μονοπατιού Η δουλειά που εκτελεί ένας κλέφτης, όταν «κλέβει» την διαδικασία ενός εργάτη.

Θέματα υλοποίησης

Αλγόριθμος κλοπής εργασιών

Βασική αρχή

Πρώτα η δουλειά (work-first scheduling).

- Ελαχιστοποίηση της επιβάρυνσης έργου (ακόμα και) σε βάρος της επιβάρυνσης του κρίσιμου μονοπατιού.

Επιβάρυνση έργου Η επιπλέον δουλειά που καλείται να κάνει κάποιος εργάτης ως μέρος του πρωτοκόλλου χρονοδρομολόγησης (spawn, sync, abort).

Επιβάρυνση κρίσιμου μονοπατιού Η δουλειά που εκτελεί ένας κλέφτης, όταν «κλέβει» την διαδικασία ενός εργάτη.

Πιο βασική αρχή

Κάνε την πιο συνηθισμένη περίπτωση όσο πιο αποδοτική γίνεται.

Θέματα υλοποίησης

Αλγόριθμος κλοπής εργασιών (συνέχεια)

- Κάθε ουρά εργασίας βρίσκεται σε διαμοιραζόμενη (συνεχή) μνήμη και μοιράζεται από όλους τους εργάτες.
- Με κάθε ουρά συσχετίζονται δύο δείκτες (indices):
  - Ο δείκτης H που δείχνει την κεφαλή της ουράς.
  - Ο δείκτης T που δείχνει το τέλος της ουράς.
  - Αμετάβλητη των δεικτών: $H \leq T$.
- Κάθε εργάτης προσθέτει και αποσπά νέες διαδικασίες από το *τέλος* της ουράς.
- Κάθε κλέφτης αποσπά διαδικασίες από την *κεφαλή* της ουράς.

Θέματα υλοποίησης

Αλγόριθμος κλοπής εργασιών (συνέχεια)

Απαιτείται συγχρονισμός όταν

- Εργάτης και κλέφτης προσπαθούν να αποσπάσουν την τελευταία εγγραφή της ουράς.
- Δύο ή περισσότεροι εργάτες προσπαθούν να αποσπάσουν εργασίες από την ίδια ουρά.

Λύση χρονοδρομολογητή Cilk

- Χρήση βαρέων κλειδωμάτων από τους κλέφτες (επιβάρυνση κρίσιμου μονοπατιού).
- Χρήση ελαφρών κλειδωμάτων από τους εργάτες/θύματα (επιβάρυνση έργου).

Θέματα υλοποίησης

Αλγόριθμος κλοπής εργασιών (συνέχεια)

Εργάτης/Θύμα

```
pop() {
    T--;
    if (H > T) {
        T++;
        lock(L);
        T--;
        if (H > T) {
            T++;
            unlock(L);
            return FAILURE;
        }
        unlock(L);
    }
    return SUCCESS;
}
```

Κλέφτης

```
steal()
{
    lock(L);
    H++;
    if (H > T) {
        H--;
        unlock(L);
        return FAILURE;
    }
    unlock(L);
    return SUCCESS;
}
```

Συμπεράσματα και Σκέψεις

- Απλό και καθαρό μοντέλο προγραμματισμού.
- Αποδοτικό όταν υπάρχει μεγάλος αριθμός από εργασίες (tasks).
- Αποδοτικό όταν το πρόβλημα είναι περιορισμένο από τους υπολογισμούς (computation-bound).
- Δείχνει ιδανικό για παράλληλο υπολογισμό αναδρομικών αλγορίθμων, π.χ., αγνωστικών ως προς την κρυφή μνήμη (cache oblivious) αλγορίθμων.
- Δείχνει κάπως περιοριστικό για την απευθείας παραλληλοποίηση επιστημονικών εφαρμογών (πρβλ. OpenMP).
- Εστιασμένο σε υπολογιστικά συστήματα διαμοιραζόμενης μνήμης.