Ιόνιο Πανεπιστήμιο – Τμήμα Πληροφορικής
Παράλληλος Προγραμματισμός
2025-26

OpenMP και παράλληλες λειτουργίες reduction

(Συγχρονισμός και έλεγχος κοινών πόρων)

https://mixstef.github.io/courses/parprog/

Μ.Στεφανιδάκης

Απαιτήσεις Reduction

- Αποκλειστική προσπέλαση πόρων
  - Οι λειτουργίες reduction απαιτούν την ενημέρωση κοινόχρηστων πόρων
- Συγχρονισμός threads
  - Για την ολοκλήρωση του reduction συχνά απαιτείται συγχρονισμός των threads
- Τι παρέχει το OpenMP
  - Δήλωση και αρχικοποίηση κοινόχρηστων πόρων
  - Κρίσιμες περιοχές για αποκλειστική προσπέλαση πόρων
  - Σημεία συγχρονισμού (barrier, έμμεσα ή ρητά δηλωμένα)
  - «Συντομεύσεις» για την εύκολη χρήση του reduction

Εμβέλεια μεταβλητών στο OpenMP

- Data Scope
  - Πολύ σημαντική έννοια στον προγραμματισμό με το OpenMP
  - Τι συμβαίνει με τις μεταβλητές του σειριακού προγράμματος; τι βλέπει κάθε thread;
- Δύο βασικές κατηγορίες μεταβλητών
  - Private: κάθε thread βλέπει διαφορετική «κόπια» της μεταβλητής
  - Shared: όλα τα threads βλέπουν την ίδια μεταβλητή
    - Κοινός πόρος, πρέπει να υπάρχει έλεγχος πρόσβασης
- Όταν δεν δηλώνεται ρητά το είδος μιας μεταβλητής, το OpenMP έχει default κανόνες για αυτό

Παράλληλη περιοχή και εμβέλεια μεταβλητών

- Μέσα στην παράλληλη περιοχή
  - Οποιαδήποτε μεταβλητή δηλώνεται μέσα στην παράλληλη περιοχή θεωρείται private
    - Το ίδιο ισχύει για μεταβλητές συναρτήσεων που καλούνται μέσα από την παράλληλη περιοχη
  - Οι global μεταβλητές είναι shared
  - Μεταβλητές που δεν έχει προσδιοριστεί το είδος τους χρησιμοποιούν το default (εξ' ορισμού shared)
- Ρητός προσδιορισμός είδους εμβέλειας
  - Το #pragma omp parallel (όπως και άλλα pragmas) παίρνουν προαιρετικά clauses προσδιορισμού εμβέλειας για λίστες μεταβλητών
    - private() και shared()

```
#pragma omp parallel private(id)
```

Αρχική τιμή private μεταβλητών

- Σε μια παράλληλη περιοχή οι μεταβλητές μπορούν να είναι shared ή private
  - Οι shared μεταβλητές έχουν πάντα την ίδια και μοναδική θέση
  - Οι private μεταβλητές έχουν μια «κόπια» ανά thread της παράλλλης περιοχής
- Αρχικοποίηση private μεταβλητών
  - Δεν γίνεται!
    - Συνήθως έχουν τιμή 0 αν δεν τις αρχικοποιήσουμε εμείς (εξαρτάται πώς το λειτουργικό σύστημα αρχικοποιεί τις σελίδες μνήμης)
  - Ακόμα και αν έχει τιμή έξω από την παράλληλη περιοχή
    - Η υπάρχουσα τιμή δεν αντιγράφεται στις «κόπιες»

Firstprivate

- Firstprivate: αντιγραφή της υπάρχουσας τιμής έξω από την παράλληλη περιοχή σε κάθε κόπια της μεταβλητής

```
int main() {

  int x = 22;

  #pragma omp parallel private(x)
      {
    printf("Thread %d: my private x=%d\n",omp_get_thread_num(),x);
  }

  #pragma omp parallel firstprivate(x)
      {
    printf("Thread %d: my firstprivate x=%d\n",omp_get_thread_num(),x);
  }

  return 0;
}
```

θα τυπώσει (μάλλον) 0: η κόπια της x δεν είναι αρχικοποιημένη

θα τυπώσει 22: η αρχική τιμή αντιγράφεται στην κόπια της x

Συγχρονισμός threads

- critical construct: υλοποίηση κρίσιμων περιοχών για αμοιβαίο αποκλεισμό κατά την ενημέρωση κοινών πόρων
  - Μπορεί να περιβάλει μια περιοχή κώδικα οσοδήποτε μεγάλη – μόνο ένα thread μπορεί κάθε στιγμή να εκτελεί τον κώδικα αυτόν
  - Χρησιμοποιεί locks (mutexes) του λειτουργικού συστήματος
    - Είναι αργή διαδικασία και μειώνει τον παραλληλισμό – δεν πρέπει να χρησιμοποιείται σε κώδικα που εκτελείται πολλές φορές
  - Named/Unnamed – περιοχές με το ίδιο (ή χωρίς) όνομα θεωρούνται η ίδια κρίσιμη περιοχή
  - Κατά την είσοδο/έξοδο στην/από την κρίσιμη περιοχή τα updates σε κοινούς πόρους που προκάλεσε κάθε thread γίνονται εγγυημένα ορατά στα άλλα threads (memory flush)

```
#pragma omp critical
{
  count++;
}
```

Συγχρονισμός threads

- Υπάρχει και το atomic construct για γρήγορες ενημερώσεις με εγγυημένη ατομικότητα εκτέλεσης
  - Χρησιμοποιείται ο μηχανισμός locking του hardware, για μικρές ποσότητες (μεταβλητές) – πιο γρήγορο από το critical construct
  - Περιορισμένη λειτουργικότητα: π.χ. ... = x (read) x = ... (write) x++ ή x += ... (update) ... = x++ (capture)

```
#pragma omp atomic    // update
gsum += sum;
```

- barrier construct: συγχρονισμός όλων των threads μιας παράλληλης περιοχής

```
#pragma omp barrier
```

master vs single construct

- To master construct
  - Απλό εργαλείο για την εκτέλεση ενός κομματιού κώδικα από το master thread
  - Δεν παρέχει κανέναν συγχρονισμό με τα άλλα threads
    - Όποιο thread δεν είναι master παρακάμπτει το κομμάτι αυτό του κώδικα
- To single construct
  - Είναι ένα worksharing construct όπως το for
  - Αναθέτει την εκτέλεση του κομματιού κώδικα σε ένα οποιοδήποτε thread
  - Στο τέλος προστίθεται ένα έμμεσο barrier – όλα τα υπόλοιπα threads θα περιμένουν να ολοκληρωθεί η εργασία στο single
    - Δυνατότητα αφαίρεσης έμμεσου barrier με το nowait

OpenMP reduction

- Οι λειτουργίες reduction μπορούν να κατασκευαστούν με όσα constructs ξέρουμε μέχρι τώρα
  - parallel for για τον κατανεμημένο υπολογισμό επιμέρους αποτελεσμάτων
    - Θα χρειαστούμε private ή firstprivate μεταβλητές
  - critical για ενημέρωση του συνολικού αποτελέσματος από τα επιμέρους αποτελέσματα
- Το OpenMP προσφέρει όμως μια πολύ πιο «αυτοματοποιημένη» λύση
  - Σε ορισμένα constructs (όπως το parallel ή το for) μπορεί να χρησιμοποιηθεί το clause reduction(τελεστής:λίστα μεταβλητών)

reduction clause

- reduction(τελεστής:λίστα μεταβλητών)
  - Ο τελεστής είναι ένας από τους +, -, *, &, |, ^, &&, | |, min, max
    - Μπορούν επίσης να οριστούν custom τελεστές με το #pragma omp declare reduction (σε τι τύπο δεδομένων, ποια η συνάρτηση reduction, ποια η αρχικοποίηση)
  - Για τις μεταβλητές που ορίζονται στη λίστα θα δημιουργηθούν «κόπιες» ανά thread
  - Οι «κόπιες» θα αρχικοποιηθούν στην κατάλληλη τιμή ανάλογα με τον τελεστή
    - Π.χ. στο 0 για το + και στο 1 για το *
  - Στο τέλος της εκτέλεσης του construct που ορίστηκε το reduction οι μεταβλητές συνδυάζονται σε τελικό αποτέλεσμα
    - Συνδυασμός σύμφωνα με τον επιλεγμένο τελεστή