Ιόνιο Πανεπιστήμιο – Τμήμα Πληροφορικής
Παράλληλος Προγραμματισμός
2025-26

OpenMP Tasks

(Παραλληλισμός δυναμικών αλγορίθμων)

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

Μ.Στεφανιδάκης

Παραλληλισμός δυναμικών αλγορίθμων

- Όταν δεν γνωρίζουμε πώς και σε ποιον βαθμό θα γίνει η κατανομή εργασίας
  - Όπως για παράδειγμα σε αλγορίθμους με αναδρομή
    - Δεν μπορούμε να ελέγξουμε εύκολα τη δημιουργία threads
  - Ή όταν έχουμε πάρα πολύ μεγάλο αριθμό εργασιών που πρέπει να ολοκληρωθούν
    - Δεν είναι δυνατή η δημιουργία τόσων πολλών threads
  - Ή όταν το φορτίο ανά thread δεν είναι ισοβαρές
- Για τις περιπτώσεις αυτές το OpenMP εισήγαγε την έννοια του “task”
  - Είναι μια αυτοδύναμη μονάδα εργασίας που πρέπει να ολοκληρωθεί μαζί με τα δεδομένα της
    - Ένας σταθερός αριθμός threads αναλαμβάνει την εκτέλεση πάρα πολλών tasks

Tasks: Βασική λειτουργία

- Μέσα σε μια παράλληλη περιοχή
  - Χρειάζεται για τη δημιουργία των threads
- Ο κώδικας συναντά ένα task construct
  - #pragma omp task
- Δημιουργείται ένα νέο task
  - «Πακέτο» κώδικα και δεδομένων για το νέο task
    - Τα δεδομένα είναι ένα snapshot τη στιγμή της δημιουργίας του νέου task – όταν αυτό θα εκτελεστεί τα αρχικά δεδομένα μπορεί να μην είναι τα ίδια ή να μην υπάρχουν!
- Το νέο task μπαίνει σε μια δεξαμενή (pool)
  - Απ' όπου το επιλέγει και το εκτελεί κάποιο thread αργότερα
    - Μπορεί όμως να εκτελεστεί και αμέσως
    - Ένα task είναι συνδεμένο με το thread που το ξεκίνησε – αν όμως είναι untied μπορεί να συνεχιστεί από άλλο thread

OpenMP Tasks: Ορολογία

- Όταν ένα «πατρικό» task (generating task) δημιουργεί ένα νέο «παιδί» task (child task) και το τελευταίο είναι
  - Undeferred task
    - Το πατρικό task μπορεί να συνεχίσει μόνο όταν έχει ολοκληρωθεί το νέο task – το νέο task μπορεί να εκτελεστεί όχι άμεσα και όχι μόνο από το ίδιο thread
  - Included task
    - Το νέο task δημιουργείται και εκτελείται αμέσως από το ίδιο thread πριν συνεχίσει το πατρικό task
  - Final task
    - Όλα τα παιδιά του νέου task θα είναι final και included
  - Mergeable task
    - Αν το νέο task είναι undeferred ή included τότε το OpenMP μπορεί να το εκτελέσει σειριακά (όχι ως νέο task)

Δημιουργία tasks (1)

- #pragma omp task

```
#pragma omp parallel
{
  #pragma omp single nowait
  {
    #pragma omp task
    {
      printf("Thread %d Task A\n",omp_get_thread_num());
    }

    #pragma omp task
    {
      printf("Thread %d Task B\n",omp_get_thread_num());
    }
  }
}
```

τι θα γίνει αν δεν υπάρχει το single construct;

Task A

Task B

- Συνήθως πολλά tasks (ή ένα task που θα δημιουργήσει όλα τα άλλα) δημιουργούνται σε ένα single construct

Δημιουργία tasks (2)

- Δημιουργία πολλών tasks σε ένα for loop

```
#pragma omp parallel
{
  #pragma omp single nowait
  {
    for (int i=0;i<10;i++) {
      #pragma omp task
      {
        printf("Thread %d executing task %c\n",
                omp_get_thread_num(),'A'+i);
      }
    }
  }
}
```

- Στα παραδείγματα που βλέπουμε δεν υπάρχει κανένας συγχρονισμός: τα νέα tasks θα εκτελεστούν και θα τελειώσουν ανεξάρτητα το ένα από το άλλο και από το «πατρικό» task

Έλεγχος δημιουργίας tasks

- Πρόσθετα clauses στο #pragma omp task για βελτιστοποιήσεις στη δημιουργία νέων tasks
- if(*expression*) clause
  - Εάν expression ψευδής, δημιουργείται undeferred task
    - Θα εκτελεστεί πριν το «πατρικό» task
- final(*expression*) clause
  - Εάν expression αληθής, δημιουργείται final task
    - Όλα τα παιδιά του θα είναι final και included
- mergeable clause
  - Εάν το νέο task είναι undeferred ή included μπορεί να εκτελεστεί χωρίς τη δημιουργία νέου task
    - Και χωρίς νέο περιβάλλον δεδομένων

Συγχρονισμός τερματισμού tasks

- Οι διάφοροι αλγόριθμοι απαιτούν κάποιο είδος συγχρονισμού μεταξύ των tasks
  - Π.χ. το «πατρικό» task να βεβαιώνεται ότι ολοκληρώθηκαν τα «παιδιά» του πριν το επόμενο βήμα
  - Ή ότι τελείωσαν όλα τα tasks μιας ομάδας (task group)
- Συγχρονισμός σε barriers
  - Το OpenMP εγγυάται ότι σε ρητά και έμμεσα barriers μέσα στην παράλληλη περιοχή η εκτέλεση θα προχωρήσει μόνο όταν ολοκληρωθούν όλα τα tasks που δημιουργήθηκαν πριν το barrier

Αναμονή για τερματισμό «παιδιών»

- #pragma omp taskwait

```
#pragma omp parallel
{
  #pragma omp single nowait
  {
    #pragma omp task
    {
      printf("Thread %d Task A\n",omp_get_thread_num());
    }

    #pragma omp task
    {
      printf("Thread %d Task B\n",omp_get_thread_num());
    }
    #pragma omp taskwait
  }
}
```

Task A

Task B

εδώ ξέρουμε ότι τα Task A και B έχουν τελειώσει

- Προσοχή: το taskwait περιμένει τα «παιδιά» αλλά όχι και πιθανούς απογόνους τους!

Αναμονή για τερματισμό ομάδας tasks

- #pragma omp taskgroup

```
#pragma omp parallel
{
  #pragma omp single nowait
  {
    #pragma omp taskgroup
    {
      #pragma omp task
      {
        atasks();
      }
      #pragma omp task
      {
        btasks();
      }
    }
  }
}
```

έμμεσο barrier: εδώ ξέρουμε ότι όλα τα tasks που δημιουργήθηκαν μέσα στο taskgroup έχουν τελειώσει

```
void atasks() {
  #pragma omp task
  {
    printf("task A1\n");
  }

  #pragma omp task
  {
    printf("task A2\n");
  }
}
void btasks() {
  for (int i=0;i<10;i++) {
    #pragma omp task
    {
      printf("task B%d\n",i);
    }
  }
}
```

Tasks και εμβέλεια μεταβλητών

- Μέσα σε ένα task υπάρχουν τα είδη μεταβλητών που γνωρίζουμε
  - shared: αναφορά στην εξωτερική μεταβλητή κατά τη δημιουργία του task
  - private: μη αρχικοποιημένη μεταβλητή, δημιουργείται κατά την έναρξη εκτέλεσης του task
  - firstprivate: αντίγραφο της εξωτερικής μεταβλητής που υπήρχε κατά τη δημιουργία του task

- Θα πρέπει πάντα να θυμόμαστε ότι η δημιουργία και η εκτέλεση ενός task πιθανόν να γίνουν σε διαφορετικούς χρόνους

Ορισμός εμβέλειας μεταβλητών σε tasks

- Μπορούμε να ορίσουμε ρητά το είδος των μεταβλητών στο #pragma omp task με τα clauses shared(*list*), private(*list*) και firsprivate(*list*)

- Default κανόνες
  - Μεταβλητές που είναι shared σε όλα τα constructs που περιβάλλουν το task (αρχίζοντας από το parallel construct) μέσα στο task θα είναι επίσης shared
  - Μεταβλητές που είναι private τη στιγμή δημιουργίας του νέου task θα είναι μέσα σε αυτό firstprivate

taskloop construct

- Συνδυάζει τη λειτουργικότητα (και τα προαιρετικά clauses) του #pragma omp for και του #pragma omp task
  - Είναι constuct για tasks που εκτελεί παρόμοια λειτουργία με ένα worksharing for construct
  - Κατανέμει την εργασία ενός for loop σε μια ομάδα tasks
    - Η μορφή του for loop πρέπει να είναι ίδια με εκείνη του for construct (γνωστή και σταθερή αρχή/τέλος, συγκεκριμένοι τελεστές, όχι αλληλοεξαρτήσεις δεδομένων μεταξύ επαναλήψεων...)
  - Στη λειτουργικότητα μοιάζει με το omp for αλλά είναι πιο ευέλικτο με τη χρήση tasks αντί για threads
    - Με κάπως μεγαλύτερη όμως επιβάρυνση

Παράδειγμα taskloop

```
#pragma omp parallel
{
  #pragma omp single nowait
  {
    #pragma omp taskloop
    for (int i=0;i<100;i++) {
      printf("Executing i = %d\n"i);
    }

    printf("After task creation\n";
  }
}
```

Δεν ξεχνάμε το omp single

Κάθε task θα εκτελέσει ένα μέρος των επαναλήψεων

- Προσοχή: το taskloop λειτουργεί ως έμμεσο taskgroup
  - Το «πατρικό» task θα σταματήσει την εκτέλεση στο } του for έως ότου τελειώσουν όλοι οι «απόγονοι» που δημιουργήθηκαν για (και κατά) την εκτέλεση του for
  - Αν δεν θέλουμε αυτή τη λειτουργικότητα χρησιμοποιούμε το προαιρετικό clause nogroup

Έλεγχος διαμοιρασμού εργασίας στο taskloop construct

- Σε πόσα και πόσο μεγάλα «κομμάτια» θα χωριστούν οι N επαναλήψεις του for μέσω των tasks
- Υπάρχουν δύο παράμετροι ελέγχου
  - grainsize: ελέγχει το μέγεθος κάθε «κομματιού» (chunk)
    - *min(grainsize,N) ≤ chunk size < 2 x grainsize*
  - num_tasks: ελέγχει πόσα tasks θα δημιουργηθούν
    - *min(num_tasks,N)*
  - Δεν επιτρέπεται να ορίζονται ταυτόχρονα και οι δύο παράμετροι
  - Αν δεν ορίζεται καμία παράμετρος, το OpenMP θα διαλέξει το «κατάλληλο» μέγεθος αυτόματα
    - Ανάλογα με την υλοποίηση της βιβλιοθήκης

Αλληλοεξαρτήσεις tasks: depend clause

- Μέχρι τώρα είδαμε μεθόδους συγχρονισμού tasks του τύπου “barrier”
  - Όταν έχουν σχέση «πατέρας» - «παιδί» (taskwait)
  - Όταν ανήκουν στην ίδια ομάδα (taskgroup, taskloop)
  - Συχνά ένα barrier καλύπτει «τεχνητά» την ανάγκη να τηρηθούν οι αλληλοεξαρτήσεις των δεδομένων
    - Μειώνοντας όμως την παραλληλία
- Πώς μπορούμε να εκφράσουμε αλληλοεξαρτήσεις;
  - Ότι τα δεδομένα εισόδου σε ένα task εξαρτώνται από την έξοδο ενός άλλου
  - depend clause: εφαρμόζεται προαιρετικά στο task construct
    - Προσοχή: λειτουργεί μόνο σε tasks που έχουν δημιουργηθεί από το ίδιο «πατρικό» task (siblings)

Σύνταξη depend clause (για tasks)

- *depend(dependence-type : list)*
  - list: ονόματα μεταβλητών (θέσεις στη μνήμη)
    - Μπορούν να είναι απλές μεταβλητές ή στοιχεία ενός array
    - Το OpenMP επιτρέπει ακόμα και τμήματα [:] από arrays
    - Στις λίστες πρέπει να εμφανίζονται ακριβώς ίδιες ή τελείως διαφορετικές θέσεις αποθήκευσης (όχι επικαλύψεις)
  - dependence-type: το είδος της αλληλοεξάρτησης
    - in: όταν κάποια μεταβλητή του list εμφανίζεται επίσης ως out ή inout σε άλλο task που δημιουργήθηκε προηγουμένως, τότε το νέο task εξαρτάται από το παλιό
    - out, inout: όταν κάποια μεταβλητή του list εμφανίζεται επίσης ως in, out ή inout σε άλλο task που δημιουργήθηκε προηγουμένως, τότε το νέο task εξαρτάται από το παλιό