Questions-Unit-4-Task-Decomposit...

Arnau_FIB

Paralelismo

3º Grado en Ingeniería Informática

Facultad de Informática de Barcelona (FIB)
Universidad Politécnica de Catalunya

Començat el	divendres, 14 maig 2021, 13:41
Estat	Acabat
Completat el	divendres, 14 maig 2021, 13:46
Temps emprat	4 minuts 52 segons
Qualificació	**5,50** sobre 6,00 (**92**%)

Pregunta **1**

Correcte

Puntuació 1,00 sobre 1,00

Assume the following code, using a tree recursive task decomposition to count how many positive values (including zero) are in a vector:

```
void hist_pos (int * vector, int n) {
      int n2 = n/2;
      if (n == 0) return;;
      if (n == 1) {
          if (vector[0] >= 0) {
              pos++;
          }
      }  else {
          #pragma omp task
          hist_pos (vector, n2);
          #pragma omp task
          hist_pos (vector + n2, n - n2);
      }
}

int pos = 0;
void main () {
       ....
       #pragma omp parallel
       #pragma omp single
       hist_pos (vector, n);
}
```

Is there any concurrency problem in this parallel program?

Trieu-ne una:

- Yes, there may be a deadlock when threads executing different tasks try to update variable pos in parallel.
- Yes, there may be a data race condition in the update of variable pos by different threads executing tasks in parallel. This can be solved by using «#pragma omp atomic» just before «pos++», which has less overhead than «#pragma omp critical». ✔ Correct!
- No, there is not any concurrency problem in this program; however, programmer should have used a cut-off mechanism to control task granularities.

La teva resposta és correcta.

La resposta correcta és: Yes, there may be a data race condition in the update of variable pos by different threads executing tasks in parallel. This can be solved by using «#pragma omp atomic» just before «pos++», which has less overhead than «#pragma omp critical».

Pregunta **2**

Correcte

Puntuació 1,00 sobre 1,00

Assume another version of the code counting the number of positive values in a vector, using tree recursive task decomposition:

```
int hist_pos (int * vector, int n) {
    int n2 = n/2;
    if (n < 1)
        return (vector[0] >= 0);
    else {
        #pragma omp task «clause-task-1»
        count1 = hist_pos (vector, n2);
        #pragma omp task «clause-task-2»
        count2 = hist_pos (vector + n2, n - n2);
        «statements»
    }
}

int pos = 0;
void main () {
    ...
    #pragma omp parallel
    #pragma omp single
    int result = hist_pos (vector, n);
    ...
}
```

Complete the previous code indicating which of the following statements are true:

Trieu-ne una o més:

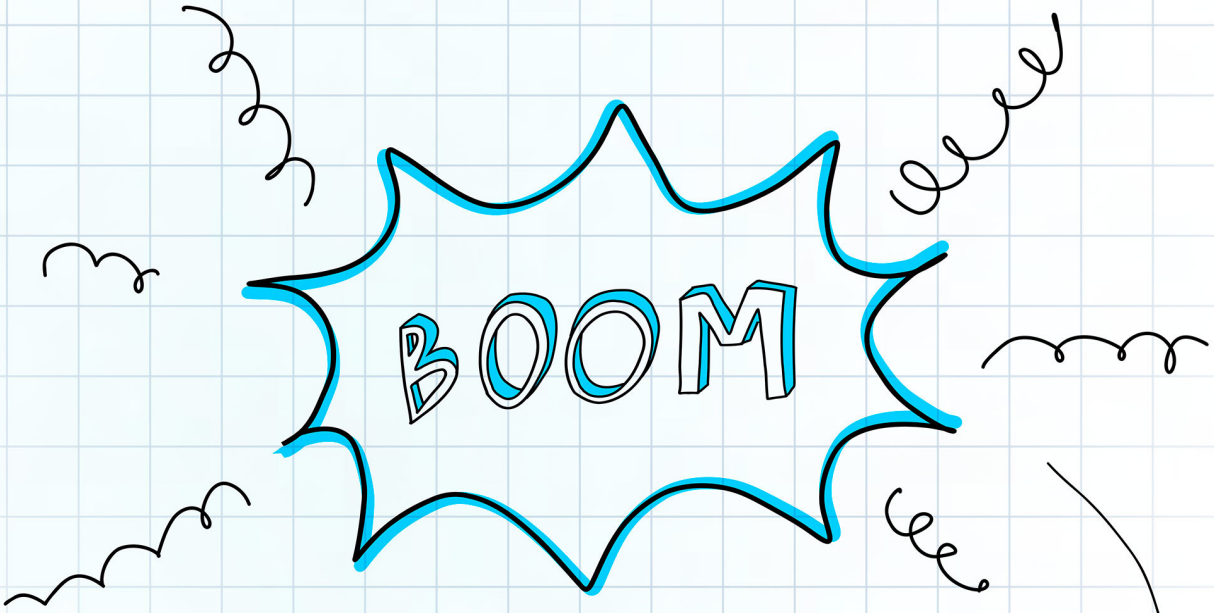
- [ ] «clause-task-1» and «clause-task-2» should specify that variables count1 and count2, respectively, are private; otherwise, the parent task and each child task may have a race condition when accessing those variables.
- [x] «clause-task-1» and «clause-task-2» should specify that variables count1 and count2 are shared, so that the parent task can see the values of count1 and count2 generated by each child task. ✔ Correct!
- [x] «statements» should include both «#pragma omp taskwait» (to wait for the two children tasks to finish and have variables count1 and count2 updated), and then «return (count1+count2)». ✔ Correct!
- [ ] «statements» only needs to include «return (count1+count2)» since tasks are executed immediately after creating them.

La teva resposta és correcta.

Les respostes correctes són: «clause-task-1» and «clause-task-2» should specify that variables count1 and count2 are shared, so that the parent task can see the values of count1 and count2 generated by each child task., «statements» should include both «#pragma omp taskwait» (to wait for the two children tasks to finish and have variables count1 and count2 updated), and then «return (count1+count2)».

WUOLAH + BBVA

1/6	BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.
Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.	



Hazte cliente de BBVA y ...
ahórrate 6 meses de suscripción

Ahora, si te abres una Cuenta Online en BBVA, te reembolsamos una de estas suscripciones durante 6 meses (hasta 9,99€/mes) al pagarla con tu tarjeta Aqua Débito

NETFLIX

HBO max

Spotify

Disney+

PlayStation Plus

DAZN

Abre tu cuenta

Promoción solo para nuevos clientes de BBVA. Válida hasta el 30/06/2023. Estas empresas no colaboran en la promoción.

Pregunta **3**

Correcte

Puntuació 1,00 sobre 1,00

Assume the following four versions of a function counting the number of positive values (including zero) in a vector (with a sufficiently large number of elements n), all versions using a linear iterative task decomposition:

```
Code 1)
int it_hist_pos (int *vector, int n) {
    int pos = 0;
    #pragma omp parallel
    #pragma omp for
    for (int i = 0; i < n; i++)
        if (vector[i] >= 0) {
            #pragma omp critical
            pos++;
        }
    return pos:
}

Code 2)
int it_hist_pos (int *vector, int n) {
    int pos = 0;
    #pragma omp parallel
    #pragma omp for
    for (int i = 0; i < n; i++)
        if (vector[i] >= 0) {
            #pragma omp critical(positives)
            pos++;
        }
    return pos:
}

Code 3)
int it_hist_pos (int *vector, int n) {
    int pos = 0;
    #pragma omp parallel
    #pragma omp for
    for (int i = 0; i < n; i++)
        if (vector[i] >= 0) {
            #pragma omp atomic
            pos++;
        }
    return pos:
}

Code 4)
int it_hist_pos (int *vector, int n) {
    int pos = 0;
    #pragma omp parallel
    #pragma omp for reduction(+: pos)
    for (int i = 0; i < n; i++)
        if (vector[i] >= 0)
            pos++;
    return pos:
}
```

Which of the following statements establishes the appropriate order of the four codes above in terms of performance, from higher to lower:

Trieu-ne una:

- Code 4 > Code 3 > Code 2 >= Code 1, taking into account both the number and kind of synchronisations performed in each case. Between Code 2 and 1 it depends on the existence of other critical regions in the program.
  ✔ Correct! Reduction only needs to protect the update of the shared variable with the contributions of per-thread local variables at the end of the loop. Atomic makes use of hardware support to protect the update of individual memory positions. The use of named critical could be better than unnamed critical if there are other parts of the program that run in parallel with this function that make use of unnamed critical regions.
- Code 3 > Code 4 > Code 2 > Code 1 since atomic incurs a negligible overhead thanks to hardware support.
- Code 2 > Code 3 > Code 4 > Code 1 since named critical regions allow a better isolation of data races, even better than atomic that only protect the read-update-write of a shared variable.
- Code 1 > Code 2 > Code 3 > Code 4 since unnamed critical has less overhead than named critical. Both kind of critical regions have less restrictions than atomic, which is also used in the implementation of the reduction clause.

La teva resposta és correcta.

La resposta correcta és: Code 4 > Code 3 > Code 2 >= Code 1, taking into account both the number and kind of synchronisations performed in each case. Between Code 2 and 1 it depends on the existence of other critical regions in the program.

Pregunta **4**

Correcte

Puntuació 1,00 sobre 1,00

Assume the following code that inserts the elements of vector ToInsert in a hash table named HashTable making use of locks to protect the access to it:

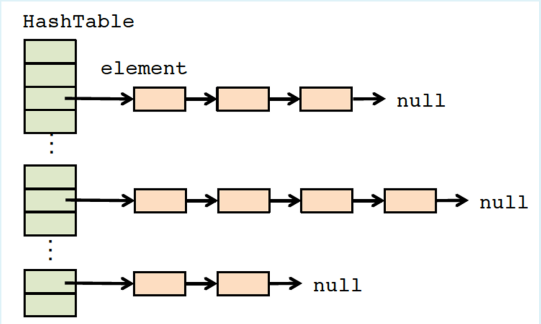
```
#define SIZE_TABLE ...
#define MAX_ELEM ...
int ToInsert[MAX_ELEM];

#define SIZE_TABLE 32384
typedef struct {
   int data;
   element * next;
} element;
element * HashTable[SIZE_TABLE];

omp_lock_t lock_vector[<<size>>];
```

```
int main() {
   ...
   #pragma omp parallel
   #pragma omp single
   for (long i = 0; i < MAX_ELEM; i++) {
      #pragma omp task <<task-clause>>
      {
          int index = hash_function(ToInsert[i], SIZE_TABLE); /* not intensive */
          <<statement-1>>
          insert_elem (ToInsert[i], index);  /* Computationally intensive task */
          <<statement-2>>
      }
   }
   ...
}
```

HashTable is defined as a vector of linked lists, as shown in the picture below:



There can not be multiple of instances of function insert_elem running in parallel for the same value of index. Which of the following statements are true?

Trieu-ne una o més:

- [x] «size» could be equal to 1. In this case «statement-1» and «statement-2» should simply set and unset, respectively, the lock variable lock_vector[0]. However, this strategy is not exploiting the whole parallelism in the parallel region. ✔ True! This solution does not allow parallel insertions in the hash table.
- [ ] «size» equal to MAX_ELEM. In this case «statement-1» and «statement-2» should simply set and unset, respectively, the lock variable lock_vector[i]; in this way each iteration of the loop can do the insertion of ToInsert[i] without any chance for data races.
- [x] «size» equal to SIZE_TABLE. In this case «statement-1» and «statement-2» should simply set and unset, respectively, the lock variable lock_vector[index]; in this way multiple iterations of the loop will compete to do insertions in the same entry of the hash table. ✔ Right! You got it!
- [x] «task-clause» could be empty. In this code, by default the compiler will capture all the variables that are needed to execute each iteration of the loop as a task. ✔ Right! variable i has to be captured, which in this case is the default.
- [ ] «task-clause» should be shared(i). By default the compiler will make variable i firstprivate; this is not the appropriate behaviour since each task needs to access to the shared variable ToInsert[i].

La teva resposta és correcta.

Les respostes correctes són: «size» could be equal to 1. In this case «statement-1» and «statement-2» should simply set and unset, respectively, the lock variable lock_vector[0]. However, this strategy is not exploiting the whole parallelism in the parallel region., «size» equal to SIZE_TABLE. In this case «statement-1» and «statement-2» should simply set and unset, respectively, the lock variable lock_vector[index]; in this way multiple iterations of the loop will compete to do

insertions in the same entry of the hash table., «task-clause» could be empty. In this code, by default the compiler will capture all the variables that are needed to execute each iteration of the loop as a task.

Pregunta **5**

Correcte

Puntuació 1,00 sobre 1,00

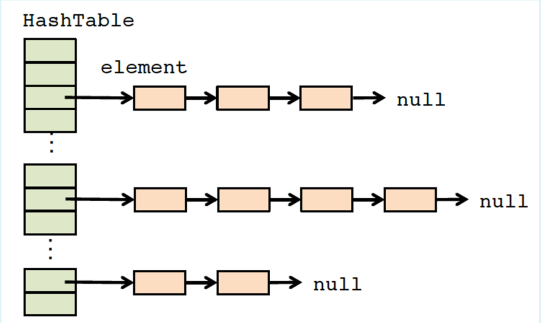
Assume the following code that inserts the elements of vector ToInsert in a hash table named HashTable making use of task dependences the access to it:

```
#define SIZE_TABLE ...
#define MAX_ELEM ...
int ToInsert[MAX_ELEM];

#define SIZE_TABLE 32384
typedef struct {
   int data;
   element * next;
} element;
element * HashTable[SIZE_TABLE];
```

```
int main() {
   ...
   #pragma omp parallel
   #pragma omp single
   for (long i = 0; i < MAX_ELEM; i++) {
      int index = hash_function(ToInsert[i], SIZE_TABLE); /* not intensive */
      #pragma omp task <<data-clause>> <<depend-clause>>
      insert_elem (ToInsert[i], index);  /* Computationally intensive task */
      }
   }
   ...
}
```

HashTable is defined as a vector of linked lists, as shown in the picture below:



Which of the following statements is true?

Trieu-ne una:

- The task definition in this code is not correct. A task should include the two statements in the loop body, i.e. the invocations of hash_function and insert_element. With that changed, «depend-clause» should be «depend(inout:index)». «data-clause» could be left empty since the defaults appropriately handle all variables used in the task.
- The task definition in this code is correct. In order to exploit maximum parallelism, «depend-clause» should be «depend(inout:index)» and «data-clause» could be left empty since the defaults appropriately handle all variables used in the task.
- The task definition in this code is correct. In order to exploit maximum parallelism, «depend-clause» should be «depend(inout:HashTable[index])» and «data-clause» could be left empty since the defaults appropriately handle all variables used in the task. ✔ Correct! You got it!

La teva resposta és correcta.

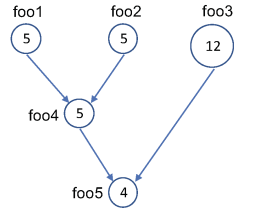
La resposta correcta és: The task definition in this code is correct. In order to exploit maximum parallelism, «depend-clause» should be «depend(inout:HashTable[index])» and «data-clause» could be left empty since the defaults appropriately handle all variables used in the task.

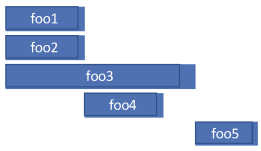
Pregunta **6**

Parcialment correcte

Puntuació 0,50 sobre 1,00

Consider the TDG (expressing dependencies between tasks) and execution timeline for the tasks on the left of the following figure:





Code 1

```
#pragma omp task
foo1()
#pragma omp task
foo2()
#pragma omp taskwait
#pragma omp task
foo3()
#pragma omp task
foo4()
#pragma omp taskwait
#pragma omp task
foo5()
```

Code 2

```
#pragma omp task
foo1()
#pragma omp task
foo2()
#pragma omp task
foo3()
#pragma omp taskwait
#pragma omp task
foo4()
#pragma omp taskwait
#pragma omp task
foo5()
```

Code 3

```
#pragma omp task
  {
  #pragma omp task
  foo1()
  #pragma omp task
  foo2()
  #pragma omp taskwait
  #pragma omp task
  foo4()
  #pragma omp taskwait
  }
#pragma omp task
foo3()
#pragma omp taskwait
#pragma omp task
foo5()
```

Code 4

```
#pragma omp task
foo3()
#pragma omp taskgroup
  {
  #pragma omp task
  foo1()
  #pragma omp task
  foo2()
  }
#pragma omp task
foo4()
#pragma omp taskwait
#pragma omp task
foo5()
```

Which of the four code above would achieve the parallelism that is available in the TDG?

Trieu-ne una o més:

- [ ] None of the codes above achieves the potential parallelism that ia available in the TDG.
- [ ] Code 1
- [ ] Code 2
- [ ] Code 3
- [x] Code 4 ✔ Correct!

La teva resposta és parcialment correcta.

Heu seleccionat correctament 1.
Les respostes correctes són: Code 3, Code 4

◄ Questions Unit 4: Task Decomposition - Implementing iterative and recursive in OpenMP

Salta a...

Lab1 laboratory assignment ►