Computación Paralela y Distribuída

Repaso Examen Parcial

Pregrado

2022-II

Profesor: José Fiestas

1.01

Indicaciones específicas:

- Esta evaluación contiene 12 páginas (incluyendo esta página) con 4 preguntas. El total de puntos son 16.
- El tiempo límite para la evaluación es 120 minutos.
- El exámen deberá ser respondida en un solo archivo pdf. Si es foto pueden ser varios archivos
- Deberá subir estos archivos directamente a https://www.gradescope.com
- Se pide activar cámaras durante el exámen. En caso no ser posible, enviar un correo de justificación

Competencias:

- Aplica conocimientos de computación apropiados para la solución de problemas definidos y sus requerimientos en la disciplina del programa. (nivel 3)
- Resuelve problemas de computación y otras disciplinas relevantes en el dominio (nivel 3)
- Analiza y valora el impacto local y global de la computación sobre las personas, las organizaciones y la sociedad (nivel 3)
- Reconoce la necesidad del aprendizaje autónomo (nivel 2)

Calificación:

Tabla de puntos (sólo para uso del professor)

Question	Points	Score
1	4	
2	4	
3	4	
4	4	
Total:	16	

1. (4 points)
Dado el siguiente código en paralelo, donde el for anidado debe ejecutarse hasta que low>high

```
#pragma omp parallel private(j)
for(i=0;i<m;i++)
{
     low=a[i];
     high=b[i];
     if(low>high)
     {
          #pragma omp single
          printf("Salida desde el for en  %d\n" ,i);
          break;
     }
#pragma omp for
for(j=low;j<high;j++)
     c[j]=(c[j]-a[i])/b[i];
}
```

¿Es una implementación correcta en paralelo? Argumente su respuesta
Respuesta: No es correcta
Si considera la implementación incorrecta, ¿Cómo corregiría el código? Proponga una forma de optimizar el código.
Respuesta: **i** debe ser privada, y se puede incluir **nowait** en el for, si **low** y **high** son privadas (optimización)

```
#pragma omp parallel private(i,j,low,high)
for(i=0;i<m;i++)
{
     low=a[i];
     high=b[i];
     if(low>high)
     {
          #pragma omp single
          printf("Salida desde el for en  %d\n" ,i);
          break;
     }
#pragma omp for nowait
for(j=low;j<high;j++)
     c[j]=(c[j]-a[i])/b[i];
}
```

La rúbrica para esta pregunta es:

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Método o algoritmo	Describe al algoritmo de solución del problema planteado en forma adecuada (2 pts)	Algoritmo con algunos errores que no afectan el resultado (1.5 pts).	Algoritmo con errores que afectan minimamente el resultado (0.5 pt).	EAlgoritmo con errores, que afectan significativamente el resultado (0 pts)
Resultados	Solución correcta usando un método adecuado (1 pt)	Errores mínimos en el método que no afectan el resultado (0.6 pts)	Errores en el método que afectan el resultado (0.3 pts)	No aplica el método ni llega a la solución correcta (0 pts).
Optimización	Solución original y optimizada (1 pt)	Solución parcialmente optimizada (0.6 pts)	Solución original pero no optimizada (0.3 pts)	Resultado encontrado no está optimizado (0 pts).

2. (4 points)
Dado el siguiente código secuencial

```
. . .
void wait(){
    x=1
    while(x==1){
        y=x+1;
        z=x+1;
    }
}
void arrive(){
    x=2;
}
int main(){
. . .
    wait();
    arrive();
. . .
}
```

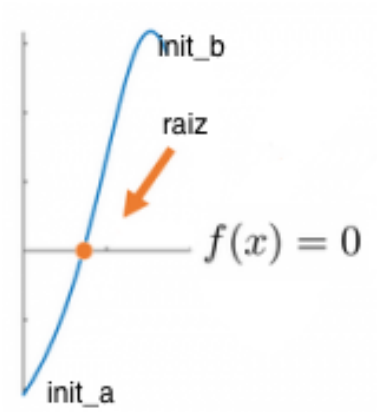
- ¿Se podrá salir del while()? **(1 pt)**
  No sale del while()
- Paralelice el codigo con OMP. Asignando una función a cada proceso. En este caso ¿Se podrá salir del while? ¿Que posibles valores tendrán x,y,z ? **(1.5 pts)**
  Se puede salir del while, en el momento en que el proceso 2 modifique el valor de x
- Si se optimiza el código (e.g. con g++ -O3 -fopenmp), ¿Se podrá salir del while? ¿Que posibles valores tendrán x,y,z ? **(1.5 pts)**
  No sale del while() porque el compilador transforma el while en infinito en la funcion wait()

Argumente cada una de sus respuestas

La rúbrica para esta pregunta es:

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Método o algoritmo	Describe al algoritmo de solución del problema planteado en forma adecuada (2 pts)	Algoritmo con algunos errores que no afectan el resultado (1.5 pts).	Algoritmo con errores que afectan minimamente el resultado (0.5 pt).	EAlgoritmo con errores, que afectan significativamente el resultado (0 pts)
Resultados	Solución correcta usando un método adecuado (1 pt)	Errores mínimos en el método que no afectan el resultado (0.6 pts)	Errores en el método que afectan el resultado (0.3 pts)	No aplica el método ni llega a la solución correcta (0 pts).
Optimización	Solución original y optimizada (1 pt)	Solución parcialmente optimizada (0.6 pts)	Solución original pero no optimizada (0.3 pts)	Resultado encontrado no está optimizado (0 pts).

3. (4 points)
3.1) Señale los constructores/cláusulas OMP necesarias para paralelizar esta funcion, que permite hayar la raíz de una función en dos dimensiones (i.e. su intersección con el eje x). **(2.5 pts)**



Input: f(función a evaluar), inita (primer valor de la funcion f), initb (último valor de la funcion f)
Output: valor raiz de f (intersección con eje x)
Procedimiento **root**:

```
. . .
act_a:= inita
act_b:= initb

for i = 0 to niters {
subint := (act_b-act_a) / nth
myleft := act_a + me * subint
myright = myleft + subint
if ( f(myleft) < 0 and f(myright) > 0) {
act_a := myleft
act_b := myright }
}
return act_a
```

Solución:

```
. . .
act_a:= inita
act_b:= initb
#pragma omp parallel
nth := omp_get_num_threads()
me := omp_get_thread_num()

for i = 0 to niters {
#pragma omp barrier
// no se necesita critical ya que solo un hilo encontrara la
    raiz de f
subint := (act_b-act_a) / nth
myleft := act_a + me * subint
myright = myleft + subint
if ( f(myleft) < 0 and f(myright) > 0) {
act_a := myleft
act_b := myright }
}
return act_a
```

3.2) Encuentre por lo menos tres errores y modifique el código para corregirlos. El codigo debe acumular el resultado de la funcion calculo en paralelo en forma eficiente **(1.5 pts)**

```
int i,tid;
double suma=0.0, suma_parcial;
#pragma omp parallel shared (part,tid) private(suma)
   reduction(*: suma)
{
     suma_parcial=0.0;
     tid=omp_get_num_threads();

     for(i=0; i<10000;i++){
          suma_parcial+= calculo(tid,i);
     }
     #pragma omp critical
          suma+=suma_parcial;
}
printf("sum: %1f\n",suma);
```

- suma debe ser compartida y se debe usar atomic en vez de critical
- part, id, i deben ser privadas

La rúbrica para esta pregunta es:

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Método o algoritmo	Describe al algoritmo de solución del problema planteado en forma adecuada (2 pts)	Algoritmo con algunos errores que no afectan el resultado (1.5 pts).	Algoritmo con errores que afectan minimamente el resultado (0.5 pt).	EAlgoritmo con errores, que afectan significativamente el resultado (0 pts)
Resultados	Solución correcta usando un método adecuado (1 pt)	Errores mínimos en el método que no afectan el resultado (0.6 pts)	Errores en el método que afectan el resultado (0.3 pts)	No aplica el método ni llega a la solución correcta (0 pts).
Optimización	Solución original y optimizada (1 pt)	Solución parcialmente optimizada (0.6 pts)	Solución original pero no optimizada (0.3 pts)	Resultado encontrado no está optimizado (0 pts).

4. (4 points)
Dado el siguiente procedimiento, donde las funciones modifican sus argumentos

```
int ejercicio3(double v[n],double x)
 {
int i,j,k=0;
double a,b,c;
a = calculo1(v,x); // n flops
b = suma_de_prefijos(v,a); // suma de prefijos de v
c = calculo3(v,x); // 4n flops
x += a*n + b + c; // calculo 4
for (i=0; i<n; i++) {  // calculo 5
    j = f(v[i],x);  // 8 flops
         if (j>0 && j<4) k++;
      }
return k;
}
```

4.1) ¿Cuál es la complejidad secuencial?

4.2) Optimice la función con directivas OMP.

4.3) Determine el speedup e indique si considera es un algoritmo escalable

Argumente sus respuestas

Solución:

```
int ejercicio3(double v[n], double x)
{
   int i, j, k = 0;
   double a, b, c;
    #pragma omp parallel
 {
   #pragma omp single {
   #pragma omp task depend(out:a)
   a = calculo1(v, x);          // n flops
   #pragma omp task depend(in:a) depend (out:b)
   b = suma_de_prefijos(v, a); // suma de prefijos de v
   #pragma omp task depend(out:c)
   c = calculo3(v, x);          // 4 n flops
   #pragma omp task depend(in:c,b,a)
   x += a * n + b + c;          /* calculo 4 */
   #pragma omp taskwait // No continua ejecucion hasta que x
       se calcule & sincroniza procesos
}
}
       /* Se usa scheduling dinamico para optimizar la
          reparticion de tareas, asimismo se reduce k, ya
          que esta acumula los resultados obtenidos en el
          condicional */
   #pragma omp parallel for schedule(dynamic) reduction(+:k)
   for (i = 0; i < n; i++)
   {
       /* calculo 5 */
       j = f(v[i], x); // 8 flops
       if (j > 0 && j <4)
k++; }
return k; }
```

Ts(n)=O(n)
Tp(n) = T_calculo1(n) + T_prefijos(n) + T_calculo3(n) + T_calculo4(1) + T_calculo5(n) = O(n), es la complejidad paralela
Aplicando Brent: Tp(n,p)= n/p+n
El speedup es,
S=Ts/Tp = n / (n/p+n) = n/(n/p+n)= 1/(1/p+1) = O(1), lo que no lo hace escalable

La rúbrica para esta pregunta es:

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Método o algoritmo	Describe al algoritmo de solución del problema planteado en forma adecuada (2 pts)	Algoritmo con algunos errores que no afectan el resultado (1.5 pts).	Algoritmo con errores que afectan minimamente el resultado (0.5 pt).	EAlgoritmo con errores, que afectan significativamente el resultado (0 pts)
Resultados	Solución correcta usando un método adecuado (1 pt)	Errores mínimos en el método que no afectan el resultado (0.6 pts)	Errores en el método que afectan el resultado (0.3 pts)	No aplica el método ni llega a la solución correcta (0 pts).
Optimización	Solución original y optimizada (1 pt)	Solución parcialmente optimizada (0.6 pts)	Solución original pero no optimizada (0.3 pts)	Resultado encontrado no está optimizado (0 pts).