
# Computacíon Paralela y Distribuída

2022-II

José Fiestas

30 de septiembre de 2022

Universidad de Ingeniería y Tecnología
jfiestas@utec.edu.pe

# Práctica Grupal Dirigida 04:
# Unidad 4: MPI/OMP
# PD04: 2 pts.(dos codigos funcionables)
# Tarea 4: 4 pts (completa, con análisis y comentarios)

# Ejercicio 1: Normalización (1 pt)

**a)** Paralelice la función de normalización con OpenMP usando dos regiones paralelas (#pragma omp parallel)

**b)** Ahora englobe a todos los bucles con solo una región paralela. En este caso, ¿tendría sentido utilizar la cláusula nowait?

```
void normaliza(double A[N][N])
        {
          int i,j;
          double suma=0.0,factor;
          for (i=0; i<N; i++) {
            for (j=0; j<N; j++) {
              suma = suma + A[i][j]*A[i][j];
} }
          factor = 1.0/sqrt(suma);
          for (i=0; i<N; i++) {
            for (j=0; j<N; j++) {
              A[i][j] = factor*A[i][j];
} }
}
```

# Ejercicio 2: Quicksort (1.5 pts)

**a)** Paralelice con OpenMP el código adjunto (qsort.c), tomando en cuenta que el algoritmo recursivo qs() del ejemplo puede ser ejecutado en tareas (tasks), dentro de un constructor parallel. Los valores iniciales a ser ordenados serán generados aleatoriamente con valores entre 1 y 99

**b)** Grafique tiempos de cálculo para N=$10^6$ vs. p=1,2,4,8, ... Haga un análisis cuantitativo del speedup y eficiencia del algoritmo y derive una expresión analítica en función a n y p para estas métricas.

**c)** Compare teoría con experimento y analice en que medida ambos coinciden (se obtiene la complejidad teorica esperada)

# Ejercicio 2: Arrays con OMP (1.5 pt)

Dada la siguiente función que realiza operaciones con los arrays A, B y C:

```
double OpArr(double A[], double B[], double C[], int n){
int i, j;
double s1, s2, a, res;
suma_de_prefijos(A,n); // obtiene el array de suma de prefijos de A
quicksort(B,n); // ordena el array B
scan_left(C+x,n);  // acumula los valores de elementos de C mas
// una constante x
for (i=0; i<n; i++) { /* primer bucle for*/
   s1=0;
   for (j=0; j<n; j++) s1+=A[j]*B[j];
   for (j=0; j<n; j++) A[j]*=s1;
}
for (i=0; i<n; i++) { /* segundo bucle for */
   s2=0;
   for (j=0; j<n; j++) s2+=B[j]*C[j];
   for (j=0; j<n; j++) C[j]*=s2;
   }
 /* calculo final */
a=s1/s2;
res=0;
for (i=0; i<n; i++) res+=a*C[i];
return res;
}
```

# Ejercicio 2: Arrays con OMP (cont.)

**a)** Identifique las tareas y sus dependencias y diagrame el DAG correspondiente

**b)** Paralelice la función con OMP. Utilice tasks con dependencias.

**c)** Grafique tiempos de cálculo para n=$10^6$ vs. p=1,2,4,8, ... Haga un análisis cuantitativo del speedup y eficiencia del algoritmo y derive una expresión analítica en función a n y p para estas métricas.

**d)** Compare teoría con experimento y analice en que medida ambos coinciden (se obtiene la complejidad teorica esperada)