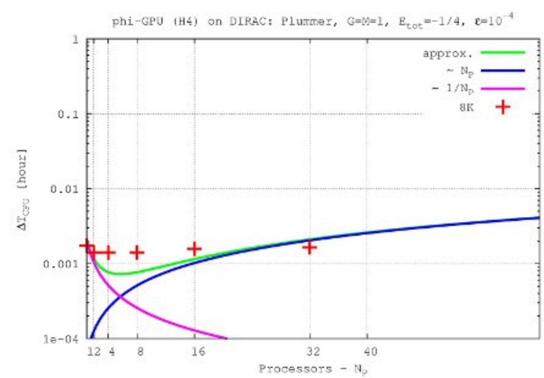
# Programación Paralela
# Modelo Exámen Parcial

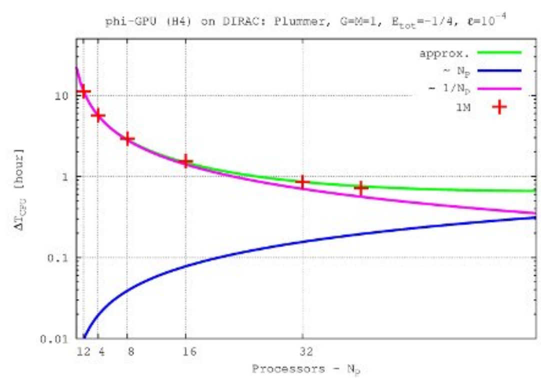

**Prof. José Fiestas**
jfiestas@uniu.edu.pe
Universidad Nacional de Ingeniería


## 1 Conceptos de algoritmos paralelos

Conteste las siguientes preguntas:

(1) Las siguientes gráficas representan tiempo vs. número de nodos de un código de N-cuerpos (izquierda: 8K cuerpos, derecha: un millón de cuerpos). Justifique la ubicación relativa de los puntos de cruce en ambas graficas y su significado ¿Cómo sería una grafica para $N = 6M$?




**Respuesta:**
Los puntos de cruce representan el número óptimo de nodos por simulación. Aprox. 6 nodos para 8K cuerpos y aprox. 64 nodos para un millón de cuerpos. Para $N = 2000$ el punto de cruce se mueve abajo y a la izquierda con respecto a los mostrados. I.e. punto óptimo de menor número de procesadores

(2) Un código de N-cuerpos llega a una velocidad teórica en FLOPs, de aproximadamente

$$S \approx \frac{\gamma N^{2+x}}{\alpha N^{2+x}/np + \beta * \log np} \tag{1}$$

donde $\gamma = 500$ es el número de operaciones de coma flotante (FLOP) por partícula, por unidad de tiempo, $\alpha = 10^{-9}$, y $\beta = 1$ son constantes de hardware, $np$ es el número de procesos, y $x = 0.31$ es una constante experimental. Calcule los FLOPs teóricos de una aplicación de N-cuerpos en un supercomputador de 5000 nodos.
Describa el comportamiento de S vs np, y la escalabilidad

**(2 puntos)**

**Respuesta:** Los FLOPs teoricos serian

$$S \approx \frac{500 \cdot N^{2.31}}{10^{-9} \cdot N^{2.31}/5000 + \log 5000} \tag{2}$$

Con esta ecuacion podemos contruir la grafica FLOPs vs np, y pòdemos tambien comparar el resultado con la grafica S (speedup) vs np
La grafica S vs np es mucho mas facil de utilizar para deducir la eficiencia E=S/np, y la escalabildad

(3) Ordene los siguientes algoritmos de acuerdo al costo

$$T_1(n,p) = \frac{n^2}{ln(n^{2/4})}, \quad T_2(n,p) = n\ ln(n^{3/2}), \quad ,T_3(n,p) = \frac{3}{2}n^{3/2} \tag{3}$$

Si $T(n,1) = O(n)$ para los 3 algoritmos, ordenelos de acuerdo a su velocidad y eficiencia $E(p) = S(p)/p$
(**2 puntos**)

**Respuesta:** $T_3 < T_2 < T_1$ La velocidad esta dada por $S_1(p) = \frac{T_{\text{seq}}}{T(p)} = \frac{\Theta(n)}{\Theta(\frac{n^2}{ln(n^{2/4})})}$
La eficiencia es $E(p) = \frac{S(p)}{p}$ Entonces: $E_3 < E_2 < E_1$

(4) Diseñe un seudocódigo en memoria distribuída que resuelva una integral de la forma $\int_a^b f(x)dx$ en n intervalos y p procesos.
- Si n es divisible entre p
- Si n no es divisible entre p

**Respuesta**
Los pasos del pseudocódigo serían:
Si n es divisible entre p:
- Distribuir intervalo de integración y número de puntos a todos los procesos

- Cada nodo realiza la sumatoria de su parte del dominio. Se itera desde rank$\cdot\frac{n}{np}$ hasta ((rank+1)n-1)/np en cada nodo. Se puede aplicar aqui varios métodos (e.g. Regla de Simpson)
- El nodo maestro reduce los datos si es requerido a una suma total.

Si n no es divisible entre p:

- complementar n para que sea divisible entre p
- asignar una cantidad variable de n en el ultimo proceso

(5) En el siguiente código

```
for (i=0; i<m; i++) {
  for (j=0; j<n; j++) {
    w[j] = procesar(j, n, v);
  }
  for (j=0; j<n; j++) {
    v[j] = w[j];
  }
}
```

```
double procesar(int j, int n, double *v);
```

Siendo n la dimension de los vectores v y w, y n es ademas la cantidad de procesos, de nombre p. El vector v está inicializado en p=0
a) ¿Cuál es la complejidad si la función procesar tiene una complejidad O(2n)?
O(m($2n^2 + n$)) = O($2mn^2$)
b) ¿Cuales serían operaciones de comunicación necesarias para paralelizar el codigo? No necesita re-escribir la función procesar()

a) Operacion de Broadcast que envíe el vector v a cada proceso
MPI_Bcast(v,n,MPI_DOUBLE,0,MPI_COMM_WORLD);
Operación de concatenación de resultados de nuevo en v
for(i=0;i<m;i++){
int a= procesar(p,n,v);
MPI_Allgather(&a,1,MPI_INT,v,1,MPI_INT,MPI_COMM_WORLD);
}

(6) Diagrame el DAG correspondiente al siguiente código

```
double funcion()
{
   int i,n,j;
   double *v,*w,*z,sv,sw,x,res;



   /* Leer los vectores v, w, z, de dimension n */
   leer(&n, &v, &w, &z);

   calcula_v(n,v);              /* tarea 1 */
   calcula_w(n,w);              /* tarea 2 */
   calcula_z(n,z);              /* tarea 3 */

   /* tarea 4 */
   for (j=0; j<n; j++) {
      sv = 0;
      for (i=0; i<n; i++) sv = sv + v[i]*w[i];
      for (i=0; i<n; i++) v[i]=sv*v[i];
   }

   /* tarea 5 */
   for (j=0; j<n; j++) {
      sw = 0;
      for (i=0; i<n; i++) sw = sw + w[i]*z[i];
      for (i=0; i<n; i++) z[i]=sw*z[i];
   }

   /* tarea 6 */
   x = sv+sw;
   for (i=0; i<n; i++) res = res+x*z[i];

   return res;
}
```