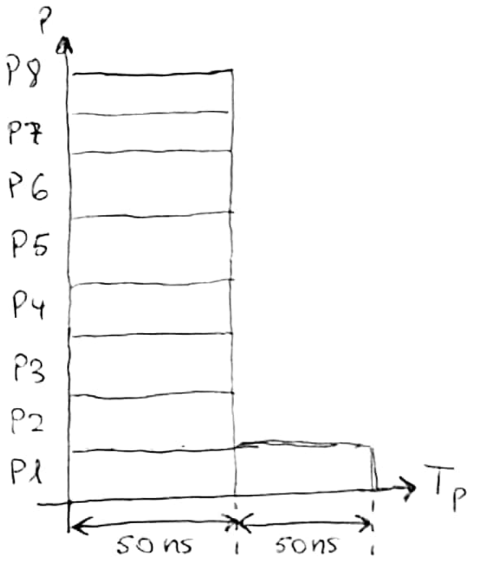
Ejercicios restantes AC: Relación 2

$\boxed{3}$ $T_p(8) = 100\,ns$



$$f_p = \frac{8 \cdot 50\,ns}{T_s} = \frac{8 \cdot 50\,ns}{50\,ns + 8 \cdot 50\,ns} = \frac{8}{9}$$

$\boxed{11}$

a)

```
int xr= sqrt(x);
if( xr > NP[M-1]){
    print("La raiz cuadrada supera la del máximo primo a detectar");
    exit(1);
}
~~bool esPrimo = false;~~
~~for(int i=0; i<N; ++i){~~
    if&
for(int i=0; ((NP[i]<xr)&&(x%NP[i])); ++i){};
if(x% NP[i]) printf("%u es primo", x);
else{ printf("%u no es primo", x);
```

b) Ejercicio 11.

```
if(idproc==0){
   for(i=1; i<num_procesos ; i++){
        send(NP, M, tipo, i, grupo);
        send(x, 1, tipo, i, grupo);
   }
}else{
   receive(NP, M, tipo, 0, grupo);
   receive(x, 1, tipo, 0, grupo);
}

i=idproc ; xr =sqrt(x) ; NP[M+i] = xr+1;
while ((NP[i] <= xr)&&(x%NP[i])) do{
     i=i+num_procesos;
}
b =(NP[i] > xr)? 1 : 0;

if (idproc==0){
   for(i=1; i<num_procesos; ++i){
      receive(baux, 1, tipo, i, grupo);
      b=b&&baux;
   }
}else send(b, 1, tipo, 0, grupo);

if (idproc==0){
   if (b) printf("%u es primo", x);
   else printf("%u no es primo", x);
}.
```

12

```
// x no es mayor que MAX_INPUT
if (x > MAX_INPUT){
   printf("%u es mayor que el máximo valor disponible", &x);
   exit(1)
}
// difusión del vector NP y de x
  broadcast(NP, M, tipo, 0, grupo);
  broadcast(x, 1, tipo, 0, grupo);
// Cálculo paralelo, asignación estática.
  xr = sqrt(x); i = idproc; NP[M+i] = xr+1;
  while((NP[i] <= xr) && (x % NP[i])){
     i = i+num_procesos;
  }
  b = (NP[i] > xr) ? 1 : 0;
// Comunicación de resultados
  reduction(b, b, 1, tipo, AND, 0, grupo);
// Proceso 0 imprime el resultado
 if(idproc == 0){
   if(b) printf("%u es primo", &x);
   else printf("%u no es primo", &x);
 }
```

b) Tiene una estructura Master-Slave (o granja de tareas) porque hay un proceso (en nuestro caso el 0) que reparte las tareas entre los demás procesos y luego los obtiene y combina para obtener un resultado.

13

a)

```c
xr = sqrt(x);
if (xr>NP[M-1]){
    printf("La raiz cuadrada de %u supera la del maximo primo a detectar
        (%u) \n", xr ,NP[M-1]);
    exit(1);
}

b=1;
#pragma omp parallel for reduction(&&:b)
for (i=0;i<M;i++){
    b = b && (x % NP[i]);
}
if (b) printf("%u es primo", x);
else   printf("%u no es primo", x);
```

b)

El orden de complejidad es $O(\frac{\lfloor\sqrt{MAX\_INPUT}\rfloor}{num\_threads \cdot L}$

Cuestiones

Cuestión 1. Indique las diferencias entre OpenMP y MPI.

- OpenMP está orientado a la programación paralela con el estilo de variables compartidas y MPI está orientado a la programación con el estilo de paso de mensajes.
- OpenMP es una API basada en directivas del compilador y funciones mientras que MPI es una API basada en funciones de biblioteca.

Cuestión 2. Ventajas e inconvenientes de una asignación estática de tareas a flujos (procesos/threads) frente a una asignación dinámica.

- Ventajas:
  - La asignación estática requiere menos instrucciones extra que la dinámica.
  - Elimina la comunicación/sincronización necesaria para asignar tareas a flujos durante la ejecución.
- Inconvenientes:
  - No se pueden utilizar si no existe un momento antes o durante la ejecución de un programa en el que se sepa con seguridad el número de tareas en total que se deben ejecutar; es decir, las tareas no aparecen todas a la vez.
  - Difícil conseguir un equilibrado de la carga cuando la plataforma (hardware/software) es heterogénea y/o no uniforme.

Cuestión 3. ¿Qué se entiende por escalabilidad lineal y por escalabilidad superlineal? Indique las causas por las que se puede obtener una escalabilidad lineal.

Para obtener una escalabilidad lineal se debe obtener una ganancia en prestaciones conforme se añaden recursos igual al número de recursos utilizados. Representa la escalabilidad ideal que se esperaría conseguir.

Para obtener una ganancia superlineal el código paralelo tiene una curva de ganancia por encima de la lineal.

La escalabilidad superlineal se puede deber al hardware y/o al código que ejecuta. Al añadir un nuevo recurso realmente en la práctica se añaden varios recursos de distinto tipo. También se puede explicar por la aplicación que se ejecuta; por ejemplo, hay aplicaciones que consisten en explorar una serie de posibilidades para encontrar una solución al problema. La exploración en paralelo puede hacer que se llegue antes a comprobar la posibilidad que lleva a una solución.

Cuestión 5. Deduzca la expresión matemática que se suele utilizar para caracterizar la ganancia escalable. Defina claramente y sin ambigüedad el punto de partida que va a utilizar para deducir esta expresión y cada una de las etiquetas que utilice.

- Punto de partida:

Se parte de un modelo de código en el que el tiempo de ejecución secuencial no permanece constante al variar el número de procesadores, lo que permanece constante es el tiempo de ejecución paralelo y en el que hay:

- Un trozo no paralelizable.
- Otro trozo (el resto) que se puede paralelizar repartiéndolo por igual entre los procesadores disponibles.

• Deducción:

La ganancia en prestaciones para este modelo de código ideal sería:

$$S(p) = \frac{T_S\,(n = kp)}{T_p} = \frac{f \times T_p + (1-f) \times T_p \times p}{T_p} = f + (1-f) \times p$$

donde:

T_S: tiempo secuencial

p: número de procesadores

T_p: tiempo paralelo

f: fracción no paralelizable.

Según esta expresión la ganancia crece de forma lineal conforme varía p con una pendiente constante de $(1-f)$.

Cuestión 6. Deduzca la expresión que caracteriza a la ley de Amdahl. Defina claramente el punto de partida y todas las etiquetas que utilice.

- Punto de partida:

  Se parte de un modelo de código secuencial en el que se verifica lo siguiente:

  - Tiene una parte no paralelizable que permanece constante aunque varíe el número de procesadores (p) disponibles. El resto se puede paralelizar con un grado de paralelismo ilimitado y, además, repartiéndolo por igual entre los p procesadores disponibles.
  - El tiempo de ejecución secuencial permanece constante aunque varíe el número de procesadores disponibles (p).

- Deducción:

  La ganancia en prestaciones para este modelo de código ideal sería (suponiendo 0 el tiempo de sobrecarga):

$$S(p) = \frac{T_S}{T_P(p)} = \frac{T_S}{f \times T_S + \frac{(1-f) \times T_S}{p}} = \frac{1}{f + \frac{(1-f)}{p}} = \frac{p}{f\,p + (1-f)}$$

$$= \frac{p}{1 + f(p-1)}$$

Scanned by CamScanner