Algorítmica

grado en ingeniería informática

Práctica 3

El supercomputador

Autores

María Jesús López Salmerón

Nazaret Román Guerrero

Laura Hernández Muñoz

José Baena Cobos

Carlos Sánchez Páez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2017-2018

Índice

Índice de figuras

1. Descripción de la práctica

El objetivo de esta práctica es diseñar un algoritmo para resolver el problema de *el supercomputador*. Este problema consiste en lo siguiente:

Descripción 1 (Supercomputador) *Una empresa debe realizar una tarea costosa: para eso tienen un supercomputador y un número ilimitado (a efectos prácticos) de computadores personales.*

Para llevar a cabo la tarea, ésta se divide en ***n*** *subprocesos que pueden llevarse a cabo de manera independiente. Cada subproceso consta de dos etapas independientes: la primera se lleva a cabo en el supercomputador con un tiempo* ***p(i)*** *segundos y la segunda etapa se lleva a cabo en uno de los computadores independientes con un tiempo de* ***f(i)*** *segundos.*

Puesto que siempre hay algún computador personal, la segunda etapa puede realizarse en paralelo. Sin embargo, el supercomputador solo puede llevar a cabo una tarea a la vez.

Se desea encontrar el orden en el que computar los subprocesos de forma que se minimice el tiempo del proceso global. Para ello se diseñará un algoritmo *voraz*.

$$\text{Tiempo de } proceso_i = \begin{cases} p(i) & \text{segundos en el supercomputador} \\ & + \\ f(i) & \text{segundos en un PC} \end{cases}$$

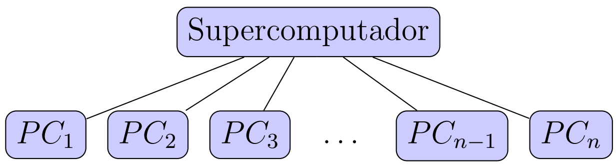


Figura 1: Descripción del problema

2. Algoritmo diseñado

Nuestro algoritmo voraz se compone de:

- **Conjunto de candidatos**. Todos los procesos a ejecutar. $P = \{p_1, p_2, ..., p_n\}$.
- **Conjunto de seleccionados**. Aquellos procesos que iremos incorporando a la lista final.
- **Función solución**. p_i completado $\forall i \in [1, \#P]$.
- **Función de factibilidad**. El tiempo de ejecución de un proceso debe ser finito.
- **Función de selección**. Seleccionaremos aquel proceso que tenga un $f(i)$ mayor.
- **Función objetivo**. Obtener la solución cuyo tiempo global sea menor, siendo $t_{fin_{global}} = t_0 + \sum_{i=1}^{n} p(i) + \max\{t_{restante}(p_1), ..., t_{restante}(p_n)\}$.

2.1. Demostración de optimalidad

Sea $P = \{p_1, p_2, ..., p_n\}$ el conjunto de procesos candidatos. Queremos demostrar que para que el tiempo global de ejecución sea mínimo debemos incorporar los elementos de P a S(conjunto solución) en orden decreciente de $f(i)$. Vamos a demostrarlo por reducción al absurdo.

Una aclaración trivial es que el tiempo que tarda cada proceso es constante. Es decir, da igual en qué momento ejecutemos un determinado p_i, ya que $t_i = p(i) + f(i)$ siempre. Lo que variará será el tiempo de ejecución global, ya que tendremos que esperar a que finalice la parte paralelizable ($f(i)$) de todos los procesos.

Sea $t_{fin_{global}} = t_0 + \sum_{i=1}^{n} p(i) + \max\{t_{restante}(p_1), ..., t_{restante}(p_n)\}$.

Sea $p_x \in P$ un proceso tal que $f(p_x) \geq f(p_i) \forall i \in [1, \#P]$.
Si p_x se ejecuta el primero:

$$t_{inicio}(f(p_x)) = t_0 + p(p_x)$$

$$t_{fin}(f(p_x)) = t_{inicio}(f(p_x)) + f(p_x)$$

Sea $t_{fin}(p) = t_0 + \sum_{i=1}^{n} p(i)$ el momento en el que han finalizado todos los cómputos del superordenador. Entonces:

$$t_{restante}(p_x) = t_{fin}(p_x) - t_{fin}(p)$$

Reducción al absurdo Supongamos que no eligiendo p_x como el primer proceso obtenemos una solución óptima. Entonces:

$$t'_{inicio}(f(p_x)) = t_0 + \underbrace{\sum_{i=1}^{pos_x - 1} p(p_i)}_{\text{espera a los anteriores}} + p(p_x) \rightarrow$$

$$t'_{inicio}(p_x) > t_{inicio}(p_x) \rightarrow t'_{fin}(p_x) > t_{fin}(p_x) \rightarrow t_{restante} p(x)' > t_{restante}(p_x)$$

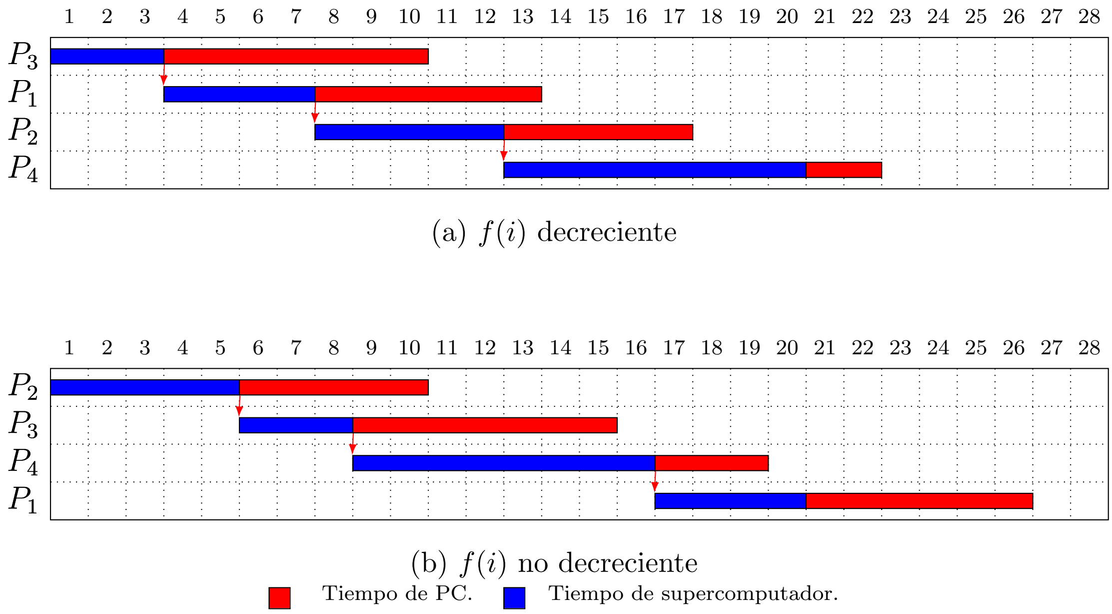
Por tanto, como $t'_{restante}(p(x))$ es mayor:

$$\max{}'\{t_{restante}(p_1), ..., t_{restante}(p_x), ..., t_{restante}(p_n)\} >$$
$$\max\{t_{restante}(p_1), ..., t_{restante}(p_x), ..., t_{restante}(p_n)\}$$

Por tanto, $t'_{fin_{global}} > t_{fin_{global}} \rightarrow$ Solución no óptima $\rightarrow$ contradicción $\rightarrow$ nuestra hipótesis es correcta.
Podemos ver este fenómeno gráficamente:

Proceso	$p(i)$	$f(i)$
P_1	4	6
P_2	5	5
P_3	3	7
P_4	8	2

Figura 2: Procesos para la demostración gráfica.



(a) $f(i)$ decreciente

(b) $f(i)$ no decreciente

Figura 3: Demostración gráfica

3. Análisis de eficiencia

Sin utilizar algoritmos voraces, una metodología para resolver el problema sería realizar todas las posibles permutaciones de procesos, calcular sus tiempos y elegir el menor. En términos de eficiencia, el problema es muy costoso, ya que obtendríamos una eficiencia de $\mathbf{O}(n!)$.

Sin embargo, si implementamos un algoritmo *voraz*, nuestro problema se reduce a ordenar los procesos en función de su $f(i)$. Por tanto, sólo necesitamos un algoritmo de ordenación (por ejemplo, *Quicksort*, con eficiencia $\mathbf{O}(nlog(n))$, significativamente mejor que la del algoritmo no voraz.

4. Conclusiones

Como hemos podido ver a lo largo de la práctica, hay ocasiones en las que un algoritmo *voraz* nos aporta una gran ventaja, ya que nos permite solucionar un problema en principio difícil y costoso de forma muy eficiente: hemos transformado un problema de cálculo de todas las posibles permutaciones de los elementos de un conjunto de entrada a un simple problema de ordenación.