PRÁCTICAS
DE
INGENIERÍA
DE LOS
COMPUTADORES

PRÁCTICA 1

INTRODUCCIÓN A LOS PROBLEMAS PARALELOS
Una aproximación del paralelismo para los problemas informáticos

Javier Bellver García (jbg39@alu.ua.es)
Alejandro Reyes Albillar (ara65@alu.ua.es)
Cesar Enrique Pozo Vásquez (cepv1@alu.ua.es)
Curso 2015 / 2016
Grado en Ingeniería Informática

Universitat d'Alacant
Universidad de Alicante

ÍNDICE

1. Introducción

Los ordenadores, o computadores personales, son una combinación de hardware y software que conforman un sistema completo. La práctica propuesta nos invita a estudiar el concepto de paralelismo aplicado a dichos ordenadores que poseen paralelismo en los niveles de hardware a nivel de componentes, nivel de circuito electrónico, nivel de lógica digital, nivel transferencia de registros y nivel sistema computador. Por encima de todos estos niveles está el nivel de sistema operativo, que actúa como interfaz entre hardware y software proporcionando una correcta transmisión de los datos a computar. A continuación se explicará brevemente qué es el paralelismo y cómo afecta al entorno tecnológico que nos rodea.

Instalaciones del superordenador Titan, en el Oak Ridge National Laboratory, en Tennesse

2. Objetivo

El objetivo principal de esta práctica consiste en adecuarse tanto al lenguaje científico como a la estructura típica de las memorias de trabajo. Además de esto, se quiere que el alumno obtenga un conocimiento actualizado sobre conceptos ampliamente difundidos relacionados con la asignatura, más específicamente con aquellos relacionados con las arquitecturas de altas prestaciones y el desarrollo de problemas paralelos. Para ello se plantean 2 preguntas:

1. ¿Para qué sirve el paralelismo en computación?
2. ¿El paralelismo puede implicar algún problema?

De este modo se deberá extraer de diferentes fuentes de información, como es internet y documentos bibliográficos a disposición del alumno, los conceptos necesarios para la correcta explicación de los conceptos y responder así a las preguntas propuestas.

3. ¿Qué es el paralelismo?

3.1 Concepto

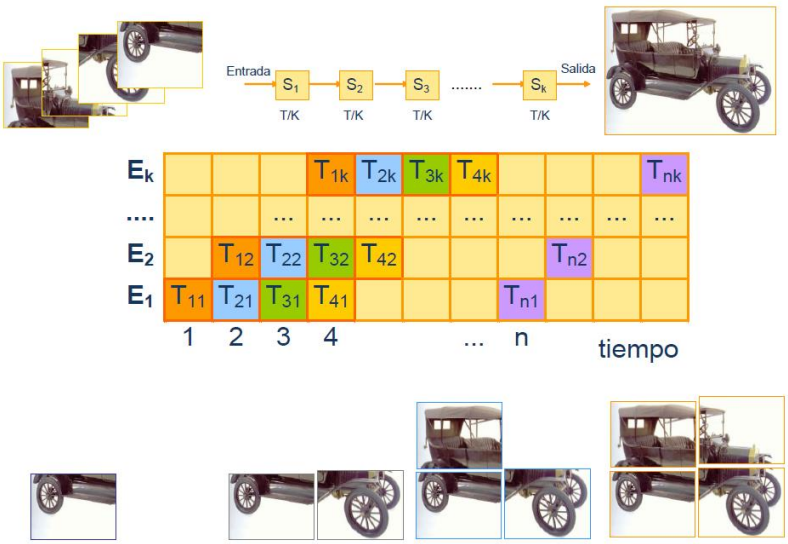
En el ámbito de la computación, decimos que existe paralelismo computacional cuando un computador ejecuta, al mismo tiempo, muchos cálculos y/o procesos, los cuales son partes de un mismo problema o programa descompuesto[1].

El paralelismo computacional nace del principio de que un gran problema complejo, **puede ser resuelto si se divide en pequeños problemas de un mismo tamaño que se resuelven al mismo tiempo**. De este modo el tiempo que se necesita para resolver un problema de gran tamaño disminuye significativamente.

El paralelismo en el ámbito del procesamiento paralelo se refiere a los aspectos relacionados con la división de una aplicación en unidades y su ejecución en múltiples procesadores para reducir tiempo y/o aumentar su complejidad.

También existe en el ámbito del procesamiento distribuido, y se refiere a la ejecución de múltiples aplicaciones al mismo tiempo con múltiples recursos situados en distintas localizaciones.

Para incrementar las prestaciones de un sistema, se aprovecha el paralelismo explícito o implícito en la entradas. Es por esto que existen dos alternativas para implementar paralelismo en un sistema y aprovechar el paralelismo de las entradas, los cuales son **replicar componentes del sistema** o **segmentar el uso de los componentes.**



Ejemplo de segmentación de una tarea

3.2. Niveles y tipos de paralelismo implícito en una aplicación:

Dentro de un código secuencial se puede encontrar paralelismo implícito en los siguientes niveles de abstracción:

- Nivel de programa: Los diferentes programas que intervienen en una aplicación o en diferentes aplicaciones se pueden ejecutar en paralelo, siendo poco probable la dependencia entre ellos.
- Nivel de funciones: Un programa puede considerarse, en un nivel bajo de abstracción, constituido por funciones. Las llamadas a funciones en un programa se pueden ejecutar en paralelo, siempre y cuando no existan ni dependencias inevitables entre ellas, ya que a mayor número de dependencias más errores podrían ocasionarse.
- Nivel de bucle (bloques): Una función puede estar basada en la ejecución de uno o varios bucles (for, while, do-while), cuyo código se ejecuta múltiples veces y en cada iteración se completa una tarea. Se pueden ejecutar en paralelo las iteraciones de un bucle, siempre y cuando se eliminen los problemas derivados de dependencias. Para detectar las dependencias hay que analizar las entradas y salidas de las iteraciones del bucle para, de esta manera, minimizar los errores debidos a ellas.
- Nivel de operaciones: En este nivel se extrae el paralelismo disponible entre operaciones. Las operaciones independientes se pueden ejecutar en paralelo. Además, en procesadores de propósito específico y propósito general se pueden encontrar instrucciones compuestas de varias operaciones que se aplican en secuencia al mismo flujo de datos de entrada.

Este paralelismo, que se puede encontrar en distintos niveles de un código secuencial, se denomina paralelismo funcional .

El paralelismo de tareas (task-parallelism) se encuentra gracias a la estructura lógica de funciones de la aplicación. En dicha estructura, los bloques funcionales, y las conexiones entre ellos reflejan el flujo de datos entre funciones, lo cual equivale al paralelismo a nivel de función dentro del código de alto nivel.[5]

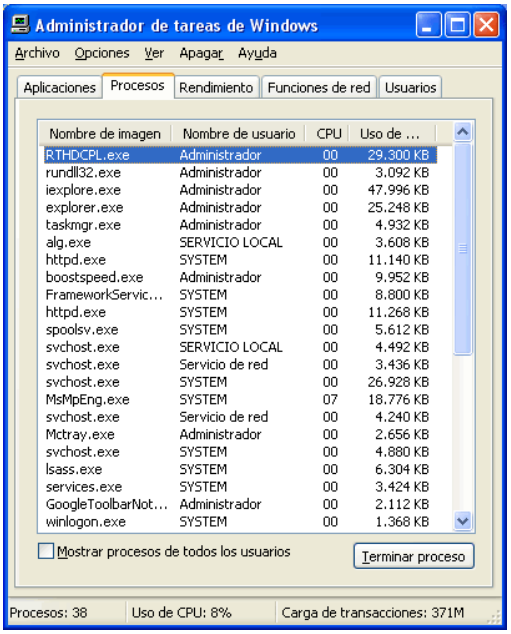
El paralelismo de datos (data-parallelism) se encuentra implícito en las operaciones con estructura de datos, tanto vectores como matrices. Se puede extraer una operación matemática de las operaciones de la aplicación. Desde los años noventa se incorporan este tipo de paralelismo en los procesadores de propósito general (MMX ,SSE, SSE2 de intel) y específicos (Trimedia de Philips).[5]

```
CPU Identification utility v2.11                (c) 1997-2013 Jan Steunebrink
-------------------------------------------------------------------------------
CPU Vendor and Model: AMD FX 8-Core series B2-rev.
Internal CPU speed  : 3110.4 MHz
System CPU count    : 1 Physical CPU(s), 8 Core(s) per CPU, 8 Thread(s)
CPU-ID Vendor string: AuthenticAMD
CPU-ID Name string  : AMD FX(tm)-8120 Eight-Core Processor
CPU-ID Signature    : 600F12
CPU Features        : Floating-Point Unit on chip  : Yes
                      Time Stamp Counter           : Yes
                      Cool'n'Quiet support         : Yes
                      Hyper-Threading Technology   : No
                      Execute Disable protection   : Yes
                      64-bit support               : Yes
                      Virtualization Technology    : Yes
Instr set extensions: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A
Size of L1 cache    : 8 x 48 KB
Integrated L2 cache : 4 x 2048 KB
Integrated L3 cache : 8192 KB
```

Información de una CPU de propósito general

3.3. Unidades de ejecución

Se conocen como unidades de ejecución a las instrucciones, hebras y procesos que componen una aplicación. El hardware se encarga de gestionar la ejecución de las instrucciones y a nivel superior el sistema operativo se encarga de gestionar la ejecución de unidades de mayor granularidad, procesos y hebras. Los sistemas operativos multihebra permiten que un proceso se componga de una o varias hebras (hilos)[5].



Administrador de sistema windows (se observa los procesos sistema operativo)

3.4. Relación entre paralelismo implícito, explícito y arquitecturas paralelas:

Los distintos niveles en los que se puede organizar una aplicación, algunos de los cuales poseen paralelismo implícito y otros que pueden hacerlo explícito, se interconectan entre sí gracias a arquitecturas paralelas que aprovechan el paralelismo. El paralelismo entre programas se utiliza a nivel de procesos. El paralelismo disponible entre funciones se puede extraer para utilizarlo a nivel de procesos o de hebras. El paralelismo dentro de un bucle se puede hacer explícito dentro de una instrucción vectorial de modo que sea aprovechado por arquitecturas SIMD o vectoriales. El paralelismo entre operaciones se puede aprovechar en arquitecturas con paralelismo a nivel de instrucción (ILP) ejecutando en paralelo las instrucciones asociadas a estas operaciones independientes[5].

4. ¿Para qué sirve el paralelismo?

Basándonos en la introducción de este trabajo el paralelismo trata de separar una misma tarea en otras subtareas que se realizan paralelamente. Como muchos avances en las ciencias de la computación, este se dio por la búsqueda de obtener un mayor rendimiento en este campo pues en teoría, al separar una tarea en diferentes procesos, podríamos acelerar la tarea en disposición al número de "subtareas" que tenemos trabajando en paralelo.

El interés por la computación paralela comenzó a finales de los años 50. Así durante los 60 y 70 se usó la computación paralela en la supercomputación, mediante supercomputadores multiprocesador que usaban memoria compartida para trabajar sobre datos compartidos.

A partir de ahí la computación paralela en supercomputación fue dominada por los MPPs, sistemas masivos de procesadores de fábrica, pero estos fueron reemplazados a mitades de los 80 por Clusters, sistemas formados por pc's enteros salidos de fábrica que usan redes estándar para la conexión[6].

En la actualidad, es normal el uso de procesadores multicore en computadores de uso cotidiano. Esto es así ya que la paralelización a nivel de hardware es más energéticamente eficiente que aumentar los ciclos de reloj en un sólo núcleo.

Dejando a un lado las mejoras en el hardware, gracias a la paralelización del trabajo, esta hace posible la implementación de muchos algoritmos que serían inviables sin la computación paralela. Un buen ejemplo de estos son los algoritmos de renderizado de gráficos, de los cuales uno es el llamado "ray tracing", que genera una imagen siguiendo el camino de la luz en la escena y simulando su interacción con los elementos de esta. En general, los algoritmos de renderizado

tienden a ser altamente paralelizables, hasta el punto en que los computadores modernos poseen GPUs que están hechas específicamente para estas tareas y que serían demasiado costosas computacionalmente para la ejecución en serie.

4.1. Aplicaciones que requieren de la computación paralela

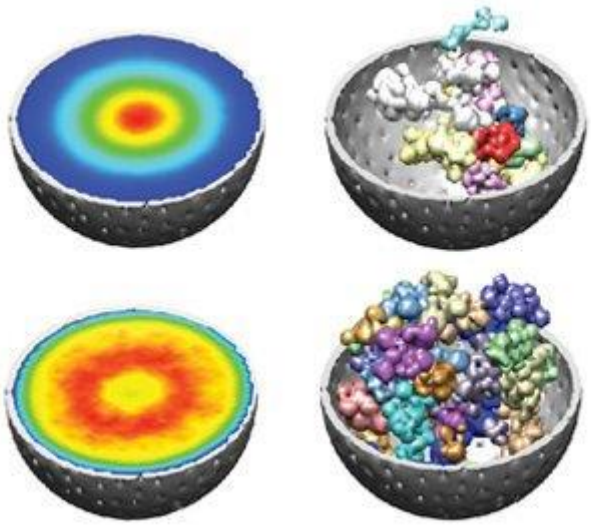
Se pueden destacar varios tipos de aplicaciones que necesitan sistemas con múltiples procesadores para su correcta ejecución:

Aplicaciones que requieren una potencia mayor que la que proporciona un sistema uniprocesador:

Pueden necesitar esta potencia para ejecutarse en un tiempo o con una calidad aceptable. Por lo que, para incrementar la potencia de cálculo con el fin de cubrir las necesidades de dichas aplicaciones, se usan múltiples procesadores, lo cual viene a ser un computador paralelo. Algunos ejemplos de estas aplicaciones son:

- **Grandes sistemas de bases de datos, servidores de aplicaciones o servidores de internet.** Estos sistemas se caracterizan por manejar una gran cantidad de datos y realizar una gran cantidad de operaciones para atender a las peticiones en un tiempo aceptable para los usuarios. Por ejemplo servidores de google.[5]

- **Grandes aplicaciones científicas y de ingeniería.** Sistemas que requieren una respuesta en un tiempo y calidad aceptables. Por ejemplo predicción meteorológica, modelado de seísmos, estudios del genoma, modelado de los océanos, química computacional, simulación de la evolución de las galaxias o simulación del daño en el impacto de vehículos.[5]



modelado del genoma humano

- **Tratamiento de imágenes y de gráficos en general.** Por ejemplo efectos especiales en películas o en aplicaciones de software para renderizar las imágenes. [5]

Juego de PlayStation®

Aplicaciones que requieren tolerancia a fallos:

Un sistema con múltiples procesadores, posee componentes redundantes, con lo que se puede conseguir que el sistema siga funcionando aunque falle uno de sus componentes. Esto es realmente imprescindible en aplicaciones críticas, ya que de fracasar, supondría una pérdida de tiempo y dinero. Por ejemplo en centrales nucleares, bases datos en empresas importantes, etc.[5]

Aplicaciones embebidas:

Son aplicaciones que se ejecutan en procesadores de bajo coste o embebidos, ya que requieren memoria compartida, memoria distribuida o ciertas prestaciones. Por ejemplo para dispositivos como cámaras digitales, DVD, videoconsolas, dispositivos móviles, etc.[5]

Smartphone

5. ¿El paralelismo puede implicar algún problema?

El paralelismo permite obtener un buen resultado en todos aquellos problemas que son paralelizables, sin embargo no todos los problemas tienen soluciones cuya resolución paralela sea óptima. Están son algunas de esas situaciones:

- **No se puede suponer que el paralelismo sea más rápido**: los resultados reales dependen de factores implicados en el rendimiento. Por esta razón es aconsejable la medición de los resultados. Por ejemplo, hay situaciones en las que un bucle secuencial es más rápido que uno paralelizado, especialmente si tiene pocas iteraciones.
- **Operaciones de escritura en ubicaciones de bloques de memoria compartida**: a diferencia de un código secuencial en el que se controla la escritura secuencialmente, en la gestión de escritura paralela puede penalizar mucho el rendimiento.
- **Paralelización excesiva**: el número de procesadores es una desventaja a la hora de paralelizar, ya que al paralelizar una tarea en muchos procesadores que tienen un enlace a un único procesador, se pierde rendimiento. En el caso de bucles anidados es mejor solo paralelizar el exterior, para no perder rendimiento. A no ser que el bucle interno sea muy largo, o si se realiza un cálculo de alto coste, se sabe que el sistema de destino tiene suficientes procesadores como para controlar el número de subprocesos que se producirán al paralelizar.

- **Llamadas a métodos no seguros para subprocesos:** la información puede resultar dañada, por problemas que son inadvertidos. Por ejemplo que múltiples procesos llamen a una misma función
- **Llamadas a métodos seguros para subprocesos:** esto ocurre debido a que la sincronización de los procesos supone un coste, si se llama a métodos seguros evitamos ese coste
- **Complicaciones entre subprocesos:** esto es debido ya que en algunos casos una tarea puede bloquear a otra.
- **Problemas con bucles como ForEach, For, ForAll:** debido a que en estos pueden existir situaciones donde una instrucción depende de otra.
- **Ejecución en paralelo de bucles en el subproceso de la interfaz de usuario**: existen situaciones en que la interfaz de usuario que se debe ejecutar en segundo plano puede resultar dañada por el uso de subprocesos, que se usan en paralelo. Por tanto puede haber interbloqueos por la espera de los subprocesos que se ejecutan en paralelo[3].

En general estos problemas específicos se pueden resumir en los siguientes problemas generales.

1ºer Problema. Sistemas no suficientemente paralelizables.

Uno de los primeros problemas que nos encontramos es el problema que surge con algunos algoritmos donde estos no son suficientemente paralelizables, de forma que la resolución paralela de estos conlleva un gran aumento en el coste computacional de este tipo de soluciones. Dos ejemplos concretos de estos algoritmos problemáticos son:

1º Encontrar el camino más corto entre dos puntos en un grafo, el problema de este algoritmo es que todas las soluciones paralelas encontradas han resultado tener un coste computacional muchísimo mayor a sus equivalentes secuenciales más óptimos[4]. Con lo cual no es un problema lo suficientemente paralelizable para que una solución paralela sea óptima.

2º La triangulación de Delaunay, nos encontramos que no existen métodos conocidos de dividir la malla en fracciones computables, simplemente no es un trabajo lo suficientemente paralelizable como para que una solución paralela sea óptima.

2ºdo Problema, distribución desigual de carga

Otro problema conocido con la computación paralela de problemas es la desigual distribución de datos debido a que de forma inherente estos algoritmos tienden a distribuir de la carga de forma desigual entre los nodos que computan el problema. Muchos algoritmos aplicados a grafos sufren de este problema, ya que al dividir el

grafo en sus nodos estos pueden estar conectados a un número variable de otros nodos, causando por tanto problemas de balanceo de carga.

3ºer problema, dificultad en la escalabilidad en los sistemas paralelos.

El último problema conocido es la difícil escalabilidad en los sistemas paralelos, esto tiene dos orígenes conocidos, el primero es la dificultad de la programación paralela donde la falta soluciones suficientemente estándares requiere que el ingeniero genere la mayor parte de soluciones a un nivel muy bajo, acabando en sistemas demasiado "únicos" como para ser fácilmente escalables. El segundo origen se da en algunos casos donde no existe una cantidad suficiente de datos reutilizables en memoria compartida provocando que el paralelismo no sea suficientemente óptimo.

6. Conclusiones:

La computación paralela está en todos los aspectos de la informática, en el nivel de arquitectura, a nivel de instrucción y a nivel software por los hilos y procesos. Se emplea cada vez en más aspectos de la informática, desde un gran ordenador que requiere un rendimiento importante o una respuesta aceptable, hasta un dispositivo embebido como puede ser un móvil. Es importante saber utilizar y manejar las herramientas que nos da la computación paralela para poder dar respuesta a los problemas de la vida diaria, para poder realizar aplicaciones que sean aceptables en tiempo, respuesta o tolerancia a fallos.

7. Bibliografía utilizada

1. Wikipedia - computacion paralela- https://en.wikipedia.org/wiki/Parallel_computingp/
2. Pagina web computing - tutorial de computacion paralela https://computing.llnl.gov/tutorials/parallel_com
3. Microsoft - https://msdn.microsoft.com/es-es/library/dd997392(v=vs.110).aspx

4. Parallel Computing Institute http://parallel.illinois.edu/blog/three-challenges-parallel-programming
5. Libro de la asignatura - Arquitectura de computadores - Julio Ortega, Mancia Anguita y Alberto Prieto, Ed. Thompson -- Tema 7- introducción computadores paralelos, programación paralela y prestaciones.

6. Intel http://www.intel.com/pressroom/kits/upcrc/ParallelComputing_backgrounder.pdf