AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

Computación Paralela for (and by) Dummies

Alario Rocio[1], Cáceres Felipe[1], Isgro Valentino[1], Ledesma Emiliano[1], Masuet Juan Pablo[1], Mazurán Clara[1], Caymes Scutari Paola[1,2], Bianchini Germán[1,2]

[1] Laboratorio de Investigación en Cómputo Paralelo Distribuido (LICPaD, UTN-FRM)
[2] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

Abstract
En este artículo se abordará el tema de la computación paralela, ya que muchos de los problemas computacionales del momento requieren del paralelismo para computar soluciones en una fracción del tiempo que tomaría computarlas en un programa secuencial. Por ello indagaremos en estos conceptos con el propósito de explicar la importancia de su aplicación. La intención del título no es de ninguna manera ofender al lector, es simplemente una advertencia de la simplicidad con la que está tratada la información, ya que somos alumnos de primer y segundo año de la facultad, además nosotros también estamos empezando a familiarizarnos con estos conceptos.

Introducción

La palabra computación proviene del latín *"computare"*, que significa "contar o calcular algo con números" [1]. Los problemas científicos e ingenieriles de la actualidad son, en su mayoría, tratados a través de una computadora, la cual a su vez se define como aquella "'Máquina electrónica capaz de realizar un tratamiento automático de la información y de resolver con gran rapidez problemas matemáticos y lógicos mediante programas informáticos" [2]. En pocas palabras, la computadora recibe entradas de información, realiza un proceso (cálculos lógicos y matemáticos) y da una salida, todo en un tiempo extremadamente breve, pero ¿será siempre este el caso?

Ingenieros, científicos e informáticos se han encontrado con problemas matemáticos tan complejos de computar, que con un solo procesador se tardarían décadas en *computar la solución*. Entre ellos el tan conocido "problema del viajante de comercio" o TSP (Travelling Salesman Problem) [3].

Supongamos que usted es un viajante de comercio y quiere vender sus productos por todo el país. Tiene una lista de 10 ciudades para visitar (sin contar la ciudad de origen) en los próximos días por lo que decide organizar el viaje para pasar por cada una de estas ciudades exactamente una vez y volver a su ciudad de origen en el camino más óptimo posible:

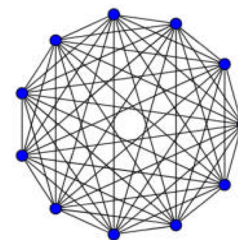

Figura 1. Representación de las distintas ciudades como nodos y los distintos caminos que se pueden recorrer como las líneas.

Como se puede observar en la Figura 1, el número de ciudades y caminos se

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

puede representar de manera gráfica a través de un grafo, en el que los vértices representan las ciudades y las aristas las posibles rutas. En este caso se considera un caso general donde todas las ciudades tienen conexión directa con el resto de las ciudades, y por eso se representa mediante un grafo completo, recalcando además que la longitud de las aristas varía y no se relaciona directamente con la longitud real de los caminos, por lo que se debe tener en cuenta que cada ruta se diferencia notablemente de demás (esto a fin de aclarar que se puede malinterpretar la figura al pensar que se puede llegar de una ciudad a otra cruzando por las rutas de los bordes).
Por lo tanto, lo que debe hacerse es calcular la cantidad de posibles caminos distintos que puede realizar. Para ello han de considerarse las distintas combinaciones en el orden para recorrer las ciudades, definiendo los tramos sucesivos.

En el primer tramo tiene 10 caminos posibles de los cuales elige uno. Luego, pasa al segundo tramo en el cual tiene que elegir entre 9 posibles caminos (una ciudad queda descartada porque ya fue visitada), elige un camino y pasa al tercer tramo en el cual tiene que elegir entre 8 caminos posibles y así sucesivamente. Es decir que la cantidad de rutas posibles a realizar es 10! = 10*9*8*7*6*5*4*3*2*1 que es igual a 3.628.800. Lo que a su vez significa a nivel computacional en cantidad y tiempo de cómputo [3].

Ahora bien, si quisiéramos hacer un algoritmo que revise y compare cada una de estas rutas, la computadora podría tardar (haciendo una burda estimación de que cada recorrido por el algoritmo tarde 0,5 segundos) 21 días en dar un resultado ¿Habrá alguna otra forma de encarar este problema con el objetivo de reducir el tiempo de cómputo, ya que no sería posible aguardar 21 días para conocer la ruta más adecuada? La respuesta es sí, y para abordar el problema eficientemente, se deberá hacer uso de las herramientas que nos brinda la computación paralela. Este artículo no se enfocará en la solución paralela a este problema específico, sino más bien será un primer acercamiento hacia el paralelismo en computación y cómo éste, desde el punto de vista teórico, es capaz de brindar herramientas a la resolución de problemas complejos. Se trata de una primera aproximación a la computación paralela de estudiantes de primer y segundo año de la carrera de Ingeniería en Sistemas de Información. En el primer semestre nos introdujimos en el tema en el marco de la asignatura Matemática Discreta y continuamos estudiándolo durante el segundo semestre en el marco de Algoritmos y Estructuras de Datos, siendo ambas asignaturas de primer nivel. Considerando el trayecto de iniciación que llevamos en nuestra formación, tomamos el desafío de volcar nuestros aprendizajes en este artículo, y es por ello que lo propusimos con este título, figuradamente.

¿Qué es la Computación Paralela?

La computación paralela es una técnica de programación en la que se ejecutan instrucciones simultáneamente. Se basa en el principio de que los problemas grandes se pueden dividir en partes más pequeñas que pueden resolverse de forma concurrente, es decir en paralelo [4].

Uno de los puntos más atractivos para la utilización de multiprocesadores es su rapidez. Muchos se preguntarán qué tan rápido es comparado al uniprocesador. El factor de aceleración, o también conocido como **Speedup factor o**, **Factor**

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

Speedup, es una medida del rendimiento relativo, definida como:

$$S(p) = \frac{t_S}{t_p}$$

En esta expresión t_s representa el tiempo de ejecución en un solo procesador y t_p es el tiempo de ejecución para un mismo problema, pero en un multiprocesador con *p* procesadores. Mediante este análisis se puede deducir que S(p) es el índice que mide el incremento en la velocidad utilizando un multiprocesador con *p* procesadores [5].

Tipos de Computadoras Paralelas y Distribución de Datos

Antes de hablar de computadoras paralelas, se debe describir brevemente a una computadora convencional. Esta consiste en una memoria principal, conectada a un procesador, como puede apreciarse en la Figura 2.

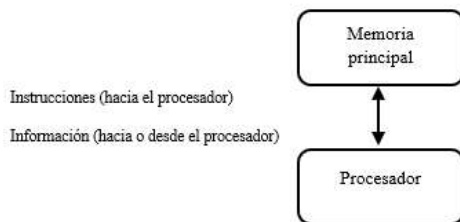

Figura 2. Estructura de una computadora convencional.

Una computadora paralela puede poseer un diseño que incluya una sola memoria con múltiples procesadores (Sistema Multiprocesador con Memoria Compartida), o múltiples memorias conectadas por una plataforma de comunicación a los procesadores (Multicomputador con Memoria Distribuida). Cuando este diseño se compone mediante varias computadoras convencionales, se lo denomina **Cluster.**

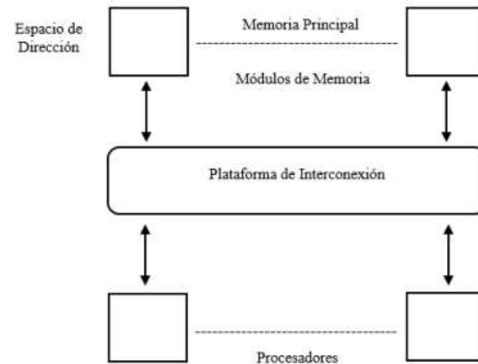

Figura 3. Estructura de un Cluster.

Un cluster consiste en la conexión de dos o más computadoras denominadas nodos con el objetivo de mejorar el rendimiento de los sistemas en la ejecución de diferentes tareas [6].

La gran ventaja que contienen los clusters se encuentra en su simpleza, ya que, son capaces de ofrecer una potencia de cómputo paralelo mediante la conexión de máquinas esencialmente secuenciales a través de una red LAN y, además, configurándolas para que puedan trabajar de forma coordinada. Muchas veces, los clusters se configuran sobre máquinas ya existentes, lo cual, no genera un gasto extra en hardware a la hora de implementar un recurso paralelo [10].

A raíz de esto, se le presenta al programador la tarea de distribuir las instrucciones y los datos a cada procesador. Para esto existen distintas formas, una de ellas sería usar un lenguaje de programación paralela de alto nivel que presente sentencias específicas para declarar variables compartidas y para distribuir

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

instrucciones entre procesadores. Esta opción no es muy popular ya que implica trabajar con un lenguaje completamente nuevo. Otra forma es utilizar una interfaz de **programación de aplicaciones** (API por sus siglas en inglés), como Open MP, que permite añadir paralelismo a programas escritos en C, C++ o Fortran. La última alternativa es crear *threads* (hilos en castellano), para trabajar en paralelo con un lenguaje de programación sin tener que recurrir a una API especializada. [5]

Para poder programar en clusters, también existen distintas herramientas que posibilitan y facilitan esta programación. Una de las herramientas es MPI (Message Passing Interface), la cual permite, en programas, comunicar datos entre procesos mediante una librería de funciones para C, C++ o Fortran [9].

Paso de mensajes

En el caso de los programas y aplicaciones paralelas/distribuidas que se ejecutan en clusters, los datos entre los procesos son intercambiados a través de mensajes entre sus procesadores. El mensaje es originado en el procesador que lo envía y se transmite a través de una red de interconexiones hacia el procesador receptor, pero este intercambio de mensajes, si no es controlado correctamente, podría ocasionar un exceso de comunicaciones y afectar a la duración del programa. Para establecer el tiempo de ejecución de un programa paralelo con *p* procesadores, se puede usar la siguiente fórmula:

$$t_p = t_{comm} + t_{comp}$$

En la cual, t_{comm} es el tiempo de comunicación y t_{comp} es el tiempo de cómputo. Cuando se divide un problema en partes que pueden ejecutarse simultáneamente, el tiempo de cómputo tiende a disminuir, ya que las partes se vuelven más pequeñas pero el tiempo de comunicación tiende a aumentar porque hay más partes que tienen que comunicarse [5].

En definitiva, el tiempo de ejecución de un programa paralelo dependerá del equilibrio entre el tiempo de comunicación y el tiempo de cómputo. En algún momento, el tiempo aumentará debido a una sobrecarga de comunicación que será esencial reducir. Esto se puede lograr buscando un punto óptimo en el cual, la cantidad de procesamiento realizado por cada procesador no entre en conflicto con el tiempo que se necesita para realizar el intercambio de mensajes entre ellos [8].

Existen dos formas de paso de mensajes: paso de mensajes con bloqueo y paso de mensajes sin bloqueo.

- ***Paso de mensaje con bloqueo (sincrónico):*** consiste en el bloqueo de procesos hasta la recepción del mensaje. El procesador transmisor del mensaje se bloquea y no permite que se siga el proceso hasta que el receptor llegue a la instrucción en la cual se recibe el mensaje y la transferencia sea completada. En esta forma de comunicación, ambos procesos tienen que estar sincronizados en el mismo punto del código para transmitir el mensaje, lo cual es más seguro, pero puede disminuir el rendimiento. Otro obstáculo de este método se produciría en el caso de que ambos procesos realicen la instrucción de “pedir al otro” al mismo tiempo, lo cual recluiría al programa en un interbloqueo o *deadlock*.

- ***Paso de mensaje sin bloqueo (asincrónico):*** en este método, el

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

proceso receptor no necesita ser "llamado" por el emisor. Es decir, cuando el transmisor llega a la instrucción de enviar el mensaje, no se bloquea para esperar al receptor: sigue ejecutando con normalidad. En cambio, si el receptor llega más rápido a la instrucción, se bloquea ya que tendría que esperar al emisor. O si incluso la recepción es no bloqueante, el receptor podría realizar otras tareas mientras aguarda la llegada del mensaje. Esta estructura de paso de mensajes es más insegura que la sincrónica (requiere mayor control por parte del programador), pero aumenta el rendimiento general del sistema [5].

Implementación, Modelo master-worker y patrones de diseño.

Desde mediados de la década de 1990, con los grandes avances tecnológicos, se comenzaron a utilizar patrones destinados al desarrollo de software. Se sabe que un patrón es una *solución reiterativa* para un problema frecuente. Pero, ¿a qué nos referimos cuando hablamos de un patrón de software?

Se considera un patrón de software a una relación de función-forma que ocurre en un contexto determinado, donde ***la función*** es descripta en términos del dominio del problema como un conjunto de cuestiones que deben resolverse para obtener un objetivo, y ***la forma*,** es una estructura descripta en términos del dominio de la solución, que busca lograr un equilibrio aceptable entre esas partes funcionales [7].

Los patrones de software se enfocan en capturar y sistematizar experiencias y técnicas satisfactorias empleadas en desarrollos de software previos, con el objetivo de crear manuales para conseguir un buen diseño y la correcta programación a la hora de realizar un nuevo desarrollo.

Sin embargo, ¿cómo podrán los patrones de software ser aplicados al cómputo paralelo?

Hay una característica fundamental de estos patrones que nos permite dar respuesta a esta pregunta. El concepto de patrón de software no está destinado para un nivel específico del diseño de software, sino que están preparados para documentar las decisiones que se tomen, respecto al desarrollo, en distintos niveles de este. Se pueden utilizar tanto como para el nivel de lenguajes de programación hasta en *un* software de sistema. Generalmente, también se utilizan en la descripción de procesos de software.

Esta gran cualidad que presentan estos patrones es esencial para poder aplicarlos al cómputo paralelo. Algunos ejemplos de aplicación son: documentar sistemas de hardware y subsistemas, mecanismos de comunicación y sincronización, políticas de particionamiento y asignación, entre otros [7].

Para poder explicar correctamente cómo se emplean los patrones en el cómputo paralelo, en este artículo vamos a mencionar un ejemplo concreto y ampliamente utilizado: ***el patrón Master-Worker*** [7].

El patrón Master-Worker es uno de los más simples utilizados en programas paralelos. Generalmente, está dedicado a resolver problemas en donde un mismo algoritmo es aplicado de forma independiente en diferentes particiones de datos.

Como su nombre lo indica, este patrón está compuesto por dos tipos de procesos. El proceso ***máster***, que

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

usualmente está asociado con el principal proceso del programa paralelo, se encarga de dividir la información entre los diferentes procesos workers. Esta información es transmitida y ejecutada en cada ***worker*** de manera simultánea. El máster espera que todos los workers terminen de hacer su trabajo y, luego, continúa haciendo su tarea.

En la Fig. 4 se muestra cómo funciona la estructura del patrón de Master-Worker.

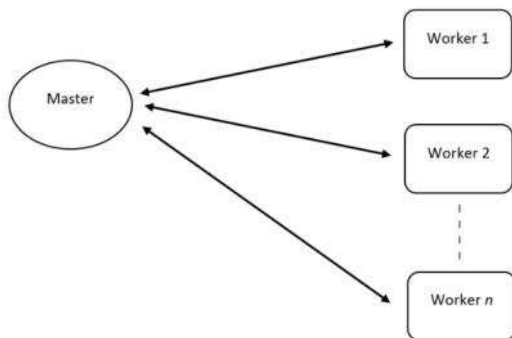

Figura 4. Diagrama de bloques de la organización del patrón Master-Worker.

Este patrón describe un tipo simple de ejecución paralela, utilizado principalmente cuando la cantidad de información que se dispone es conocida de antemano y es fácil dividirla en partes equivalentes para que la operación de los workers sobre dicha información no dependa una de la otra. Este último requisito es fundamental para el posible y correcto funcionamiento del patrón Master-Worker, ya que la ausencia de la dependencia entre las distintas particiones de la información asegura que no sea necesaria la sincronización entre los procesos workers [7].

Sin embargo, no siempre la información puede dividirse de tal manera para que los workers trabajen de manera completamente independiente, sin relacionarse entre sí. El modelo Master-Worker, en situaciones determinadas, puede trabajar con los workers comunicándose entre sí y compartiendo información entre ellos.

Escalabilidad

Otro concepto muy importante es la **escalabilidad.** Esta es la capacidad de adaptación y respuesta de un sistema respecto al rendimiento del mismo en relación a sus recursos físicos. El rendimiento depende proporcionalmente del tamaño del sistema. Cuanto más grande sea el sistema mejor debería ser el rendimiento alcanzable, o el mismo debería mantenerse ante la aplicación de problemas más complejos.

El estudio de la escalabilidad puede considerarse en términos de hardware y en términos del programa o algoritmo:

- ***Escalabilidad en Arquitectura o Hardware:*** Indica la capacidad de un diseño de hardware de ser incrementado en tamaño con el objetivo de obtener un mayor rendimiento.
- ***Escalabilidad algorítmica:*** Indica la capacidad de un algoritmo paralelo de recibir más datos manteniendo baja o proporcional la cantidad de pasos computacionales.

Por supuesto, sería deseable que todos los sistemas multiprocesadores fuesen escalables en hardware, pero esto dependerá mayormente en el diseño del sistema. No es tan sencillo como parece debido a que, al añadir procesadores al sistema, la red de interconexión debe ser expandida. Como resultado existirá mayor demora en comunicación y mayor contención de recursos. En consecuencia, la eficiencia del sistema podría llegar a reducirse.

La escalabilidad sugiere que los problemas de mayor tamaño pueden

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

adaptarse a sistemas de mayor tamaño (aumentar el tamaño del sistema significa claramente agregar procesadores). Sin embargo, duplicar el tamaño del problema no necesariamente duplicaría el número de pasos computacionales, ya que cambian dependiendo del problema.

Por ejemplo, sumar dos matrices tiene este efecto, pero multiplicar matrices no. El número de pasos computacionales para multiplicar matrices se cuadruplicaría en lugar de duplicarse. Por lo tanto, escalar diferentes problemas implicaría diferentes requisitos computacionales [5].

Conclusión

Recurrir al cómputo paralelo se hace necesario ante las limitaciones de los uniprocesadores. Retomando con el "problema del viajante de comercio" expuesto en la introducción, podríamos estimar la optimización y reducción de tiempo que podemos llegar a lograr paralelizando el algoritmo de búsqueda.

Como mencionamos anteriormente, el tiempo que se demoraría el programa en recorrer y comparar todas las posibles rutas (exactamente una vez) en un uniprocesador, con un cálculo de cada recorrido hipotético de 500 ms, es de 21 días. Por lo tanto, si aplicásemos el mismo algoritmo en un multiprocesador de 42 nodos el tiempo de cómputo se reduciría a 12 horas (esto teniendo en cuenta una escalabilidad y división del trabajo ideal, sin tiempos extra de comunicación y sincronización). Siendo este un estudio de la computación paralela desde el punto de vista teórico e introductorio, esta estimación es grosera y no considera los posibles obstáculos que representaría paralelizar un algoritmo.

No obstante, muchos de estos inconvenientes son subsanables, dado que, como se comentó en el desarrollo del artículo, existen diversas arquitecturas paralelas y casos que no resultan costosos en comparación con otro tipo de arquitecturas, e incluso pueden confeccionarse en base a equipos de uso cotidiano y usando software de libre distribución. Por su parte, las dificultades en la programación son resolubles mediante el análisis y estudio del paradigma paralelo por parte del desarrollador. Como resultado, la ganancia que puede obtenerse es significativa: como vimos en el ejemplo del viajante, el requisito inicial era lograr definir el recorrido para un inminente viaje, que no podía demorarse 21 días en comenzar. No obstante, aplicando un esquema paralelo sobre una arquitectura con 42 procesadores (por ejemplo, un cluster de 7 nodos multicore con 6 procesadores cada uno), nos podría brindar la solución en horas (quizá no 12 como la solución ideal, pero incluso duplicando este tiempo, en tan solo 24 horas tendríamos el recorrido, lo cual nos ahorraría 20 días de espera).

Para finalizar con nuestra idea consideramos que con un estudio a conciencia de las características del problema a solucionar y el conocimiento de las posibilidades que nos brinda el cómputo paralelo, la ganancia que podemos obtener de este paradigma resulta muy valiosa en la actualidad, donde cada vez aparecen problemas más complejos y los volúmenes de datos e información que surgen cada día comienzan a tornarse muy difíciles de procesar.

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

Bibliografía/Referencias

1 –Apuntes sobre los términos "computación" e "informática" http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94351994000300009. Fecha de acceso: agosto 2022

2- Diccionario panhispánico de dudas. https://www.rae.es/dpd/computador. Fecha de acceso: agosto 2022

3- https://www.youtube.com/watch?v=oSPkod-M6Gc

4- Computación Paralela. https://conceptosarquitecturadecomputadoras.wordpress.com/computacion-paralela/ Fecha de acceso: septiembre 2022

5 - Parallel Programing Wilkinson (2004) Barry Wilkinson y Michael Allen14

6- Introducción: Clústeres. https://www.ibm.com/docs/es/was-zos/9.0.5?topic=servers-introduction-clusters. Fecha de acceso: septiembre 2022

7- Patterns for Parallel Software Design (2010), Jorge Luis Ortega-Arjona.

8- https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.473.7488&rep=rep1&type=pdf

9- Introducción a MPI. http://informatica.uv.es/iiguia/ALP/materiales2005/2_2_introMPI.htm. Fecha de acceso: septiembre 2022

10- Computación Paralela: Introducción (parte I) https://youtu.be/h1zWkRdctp8. Fecha de ingreso: septiembre 2022