¿Concurrencia y Paralelismo en el primer curso de Algorítmica?

Armando De Giusti, Fernando Emmanuel Frati
Instituto de Investigación en Informática III-LIDI. Facultad de Informática
{degiusti, fefrati}@lidi.info.unlp.edu.ar

Resumen

Se presenta un tema de intensa discusión curricular en la actualidad: ¿El cambio tecnológico en los procesadores impondrá un cambio en el enfoque de la enseñanza de la programación, reemplazando el paradigma secuencial por el paralelo?
La inclusión temprana de los temas de concurrencia y paralelismo en la currícula de Informática está en discusión y en este trabajo se presenta un análisis de posibilidades, así como una propuesta en desarrollo en la UNLP.

Palabras claves: *Algoritmos, Concurrencia, Paralelismo, Programación Secuencial, Programación Paralela.*

Introducción

Agotamiento del Modelo Von Neumann

Desde el nacimiento de las computadoras en la década del 50, el aprendizaje inicial de la programación ha estado asociado con el "modelo de Von Neumann" [1] cuya simplicidad y consistencia en el manejo de control y memoria le permitió sostenerse a lo largo de más de 5 décadas.

Sin embargo, resulta claro que la evolución de la tecnología de hardware (y también de software) fueron convirtiendo el "modelo Von Neumann" en una abstracción útil para analizar "threads" de programación secuencial, pero cada vez más difícil de sostener como paradigma real, ante computadoras con múltiples procesadores que trabajan en forma paralela (incluso en una PC de escritorio). [2] [3].

La aparición de los procesadores de múltiples núcleos ("multicores") [4] ha significado el agotamiento definitivo del modelo Von Neumann, ya que los procesadores (aún en sus expresiones más simples) presentan una arquitectura esencial MIMD sin reloj único y con capacidad de procesamiento paralelo y múltiples memorias que pueden ser total o parcialmente compartidas. [5] [6] [7].

Impacto de la tecnología de Procesadores de Múltiples Núcleos

Una arquitectura típica de un procesador multicore, tal como la que se ve en la Fig. 1, presenta varios elementos a considerar:

- Cada núcleo tiene su propio reloj, memoria local y unidad aritmético lógica.
- Existen memorias compartidas globales y locales (por ej. entre dos núcleos), con lo cual el mapa global de memoria es de acceso no uniforme (NUMA).
- Una aplicación a ejecutarse sobre un procesador como el de la Fig. 1, puede resolverse como múltiples hilos de control ejecutándose concurrentemente sobre los diferentes núcleos.
- Claramente esto requiere (al menos) resolver dos problemas básicos: la descomposición y asignación de las tareas concurrentes a los núcleos y la coordinación (comunicación y sincronización) entre los núcleos para resolver la aplicación.

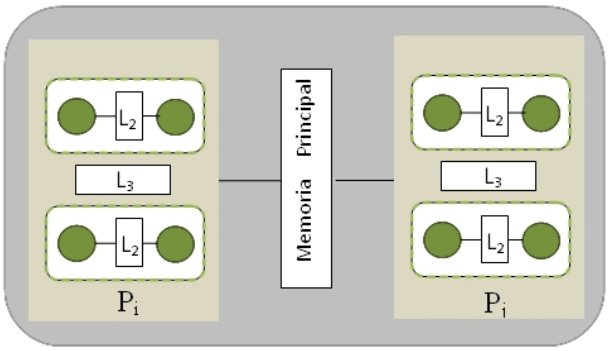

Figura 1: Diagrama de estructura de un multicore

Este breve análisis nos muestra que debemos considerar la "máquina paralela" subyacente para resolver la aplicación, lo cual impacta sobre todos los niveles de software, especialmente los lenguajes, compiladores y sistema operativo. [8]

Por otra parte las métricas clásicas de eficiencia de los algoritmos deberán considerar el número de núcleos utilizados, su rendimiento y la bondad de la descomposición concurrente que se ha realizado de la aplicación para reducir los tiempos ociosos. [9]

Los problemas del mundo real.

En este punto, debemos considerar que los problemas del mundo real son *esencialmente paralelos*, con lo cual las nuevas arquitecturas si bien son más complejas, también nos permiten soluciones que se adapten más eficientemente al modelo de las aplicaciones.

Resulta claro que hemos formado nuestros estudiantes y docentes con el "modelo Von Neumann" como fundamento para la programación secuencial (como primer paso para resolver algoritmos) y también para el desarrollo de lenguajes y compiladores.

En este punto una dificultad del "mundo real paralelo" es que resulta naturalmente más complejo… no sólo para los programadores sino también para los desarrolladores de software de base. En este punto (como es clásico) las arquitecturas de los procesadores están un paso delante de lo que las herramientas de software pueden explotar eficientemente. [10]

Cursos clásicos de Algorítmica

Un curso clásico de Algorítmica está centrado en la expresión del control mediante un número reducido de primitivas y la introducción de estructuras de datos elementales.
Normalmente se analiza un único flujo de control, con una métrica elemental de eficiencia dada por el tiempo de ejecución de un dado algoritmo para un determinado conjunto de datos de entrada.
La verificación de los algoritmos resulta bastante directa y su validación es posible respetando algunos principios básicos de la programación estructurada. [11]
Conceptos como los de modularización, documentación o reuso se incorporan como "buenas prácticas" de programación.
El análisis de los problemas se concentra en encontrar una expresión algorítmica correcta y en lo posible eficiente, para alcanzar la solución.
En todo momento subyace una arquitectura de único thread de control, con reloj único.

Desafíos actuales

Resulta claro que si asumimos el agotamiento del modelo Von Neumann, es necesario asumir múltiples desafíos en la enseñanza de la programación. En particular replantearnos un curso "clásico" de Algoritmos.
Ejecutar eficientemente una aplicación sobre una arquitectura como la de la Fig. 1 conducirá casi siempre a la programación paralela. La programación secuencial será sólo un "caso degenerado" poco aplicable, ya que conduciría a tener N-1 núcleos del procesador ociosos.

Lenguajes y Modelos de Programación

El análisis de lo expuesto en la Introducción, nos lleva a concluir que el *concepto clave* que

los alumnos deben incorporar es la *concurrencia.*

Estudiar la concurrencia significa básicamente concentrarnos en dos puntos:

- La comunicación entre procesos concurrentes (que normalmente significa una forma de *cooperación* entre ellos).
- La sincronicación entre procesos concurrentes (que normalmente significa una forma de *competencia o dependencia* entre ellos).

Resulta claro que el impacto más directo de las arquitecturas crecientemente paralelas es que los conceptos "abstractos" de concurrencia (tal como se estudiaban en los 80) se convierten en "reales" con procesos ejecutándose en procesadores (núcleos) diferentes, con mecanismos de comunicación heterogéneos que pueden ser explotados por diferentes primitivas de los lenguajes de programación.

Es interesante retomar la imagen de la Fig. 1 y comprender que UN procesador se convierte en un sistema distribuido/paralelo con memoria compartida y una red de comunicación de alta velocidad, que puede tener tiempos diferentes para comunicar diferentes pares de procesadores.
Este modelo obliga a resolver temas aún abiertos para los sistemas distribuidos/paralelos, como tener lenguajes de programación y ambientes de desarrollo de software orientados a especificar algoritmos paralelos y derivar código eficiente y validable. [12] [13]
Estudiar sincronización y comunicación entre procesos residentes en los núcleos de una arquitectura como la de la Fig. 1 significa analizar tres enfoques:

- Memoria compartida.
- Mensajes.
- Híbrido.

Si bien las supercomputadoras y los sistemas distribuidos conducen naturalmente a los dos primeros, los procesadores de múltiples núcleos obligan a considerar esquemas híbridos donde los lenguajes de expresión de algoritmos permitan utilizar tanto memoria compartida como mensajes. [14]
Esta alternativa le agrega un punto más de complejidad a los compiladores y herramientas de desarrollo de algoritmos paralelos. [15] [16]
Notemos que los esquemas de programación de sistemas paralelos [17] también están en evolución, por el cambio en los procesadores que constituyen el elemento básico de un sistema paralelo tal como un cluster.
Resumiendo el impacto de los procesadores de múltiples núcleos significa:

- La noción de "problema secuencial" que se traduce en un "algoritmo secuencial" es reemplazada por la necesidad de explotar el paralelismo explícito de la arquitectura.
- Los modelos de sincronización y comunicación entre procesos se tienen que adaptar al modelo híbrido que presentan las arquitecturas.
- Los lenguajes de programación (y por ende de expresión de algoritmos) deben incorporar las primitivas de manejo de procesos concurrentes, soportando las características propias de las nuevas arquitecturas.
- Es necesario formar recursos humanos capacitados en el análisis de problemas del mundo real, considerando el paralelismo explícito de los procesadores que van a utilizarse en su resolución.

¿Cuándo se introduce Concurrencia y Paralelismo en la Currícula actual?

En la mayoría de las carreras de Informática en Argentina los temas de Concurrencia están en el 3er. Año de la currícula. En la forma de asignaturas específicas o asociados con temas de Sistemas Operativos.
Los temas de Sistemas Paralelos y Algoritmos Paralelos normalmente se presentan más adelante, en 4to. o 5to. Año.
En alguna de las currículas aparecen aspectos de Concurrencia en 2do. año, asociados con el aprendizaje de algún lenguaje en particular o desde un análisis de los diferentes paradigmas de programación.

En estos casos, muy pocas veces se relaciona la concurrencia con la arquitectura física donde se ejecutará el algoritmo, porque el alumno ha visto en los cursos introductorios el concepto de "máquina Von Neumann".

La transición para visualizar concurrencia y paralelismo sobre una arquitectura MIMD real, normalmente se da una vez que el alumno cerró el ciclo de programación imperativa y orientada a objetos, así como los cursos básicos de Arquitectura, Redes y Sistemas Operativos.

Este enfoque "tardío" resulta difícil de sostener ante la evolución de las arquitecturas de los procesadores. Al mismo tiempo está demostrado que los conceptos de concurrencia y paralelismo requieren un esfuerzo significativo a los alumnos.

¿Qué sería deseable en un primer curso de Algorítmica?

Si aceptamos que concurrencia resuelta un concepto básico que es necesario aprender en forma temprana, de modo de incorporarlo en el ciclo formativo del alumno, sería deseable:

- Trabajar con un modelo de arquitectura que explicite el multiprocesamiento.
- Analizar enunciados de problemas que requieran la descomposición (al menos elemental) en tareas concurrentes.
- Discutir la asignación de tareas a los procesadores (núcleos) de la arquitectura básica.
- Trasmitir la visión de Hoare de un sistema concurrente/paralelo relacionada con "comunicar y sincronizar procesos secuenciales vinculados con la misma aplicación o ejecutados sobre recursos compartidos".
- Los "nuevos temas" son la comunicación y sincronización por mensajes y por memoria compartida.
- Se requerirá un lenguaje de expresión de algoritmos paralelos simples (en lo posible soportado por un ambiente visual para fortalecer el aprendizaje de los conceptos).
- Deben incorporarse trabajos experimentales que resuelvan algoritmos paralelos.

El curso de Algorítmica en la UNLP

Situación actual

El curso que se dicta actualmente en la Facultad de Informática de la UNLP abarca dos semestres y tiene el título genérico de "Algoritmos, Datos y Programas". Responde a un modelo clásico centrado en el aprendizaje de la expresión de algoritmos, la introducción a las estructuras de datos lineales y no lineales y la incorporación de conceptos relacionados con modularización, análisis de eficiencia, abstracción y reuso [18].
En el 2do. semestre se introduce conceptualmente el paradigma de objetos.
Las tareas experimentales se realizan inicialmente con un entorno propio Visual Da Vinci [19] y posteriormente con Pascal y Delphi.

Extensiones/Cambios en curso

A partir del año 2010 se ha propuesto un cambio gradual, que tiene 3 etapas:

1. Incorporación de la descripción de las arquitecturas de los nuevos procesadores, como evolución de la máquina de Von Neumann e Introducción a los conceptos básicos de Concurrencia.

2. Desarrollo de un entorno visual de expresión de Algoritmos concurrentes y paralelos que resulte en una evolución del Visual Da Vinci (VDV) y admita mecanismos de comunicación y sincronización, tanto por memoria compartida como por mensajes.

3. Rediseño del programa de la asignatura, planteando la programación concurrente/paralela como el caso general y la programación secuencial como la excepción.

El entorno LMRE.

Para el trabajo experimental con este nuevo enfoque se ha definido un entorno específico llamado LMRE (Lidi Multirobot Environment) con las siguientes características principales:

- Extiende el ambiente VDV para introducir las nociones de concurrencia, aprovechando las ventajas de utilizar una herramienta de desarrollo visual que permita al estudiante conectarse rápidamente con los desafíos del desarrollo de aplicaciones concurrentes.

- Emplea múltiples robots (que representan los procesadores del mundo real) con áreas de recorrido privado y de recorrido compartido. Se repiten los objetos simples del VDV que el robot visualiza, cuenta y recoge o deposita, ahora en coordinación con otros robots.

- Habilita (y visualiza) los mecanismos de sincronización por mensajes y por memoria compartida (sólo con semáforos explícitos en la primer versión).

- Permite retomar problemas resueltos secuencialmente y contrastarlos con una solución concurrente. Su objetivo se centra en una contribución a los esquemas de pensamiento que el estudiante adquiere durante su primer año de estudio para la resolución de problemas.

- Se espera que reduzca la curva de aprendizaje: como los estudiantes ya se encuentran habituados al lenguaje y entorno, pueden enfocarse directamente en los nuevos conceptos.

- Permite visualizar los conceptos que se quieren transmitir: extendiendo la idea del robot que se mueve por la ciudad a varios robots, es necesario reflejar visualmente espacios de sincronización y/o exclusión mutua dentro de la ciudad.

- Permite la evolución natural de estos conceptos: la herramienta debe correr en varios niveles de abstracción, de manera que puedan complicarse o simplificarse los mecanismos que establecen sincronización y/o exclusión.

Actualmente se está desarrollando el entorno, en lenguaje JAVA y partiendo del esquema largamente probado del ambiente VDV. Se estima tenerlo en prueba en el 2do. semestre de 2010.

Conclusiones y Líneas de I/D

La incorporación temprana de los temas de concurrencia y paralelismo en las currículas de Informática es inevitable, dada la evolución de la tecnología de los procesadores y también del software de base. [20]

Posiblemente la transformación de un curso de algorítmica secuencial “clásico” en un curso de algorítmica paralela requiera un tiempo de maduración, de desarrollo de herramientas específicas y también de capacitación de los docentes. El carácter normalmente masivo de los cursos iniciales de Informática (en nuestro caso del orden de 1000 alumnos) complica la implementación de estos cambios en un corto plazo.

Las líneas de investigación principales son la programación paralela con mecanismos de comunicación y sincronización “híbridos” y el desarrollo de software de base que explote el paralelismo de las arquitecturas en forma eficiente. [21]

Referencias

[1] A. W. Burks, H. H. Goldstine, and J. von Neumann, “Preliminary discussion of the logical design of an electronic computing instrument,” Papers of John von Neumann on Computing and Computer Theory, 1947.

[2] M. D. Hill and M. R. Marty, “Amdahl’s law in the multicore era,” Computer, vol. 41, pp. 33–38, 2008.

[3] A. Marowka, “Parallel computing on any desktop,” Commun. ACM, vol. 50, no. 9, pp. 74–78, 2007.

[4] J. Held, J. Bautista, and S. Koehl, “From a few cores to many: A tera-scale computing research overview,” tech. rep., Intel Corporation, 2006.

[5] D. J. Ernst and D. E. Stevenson, “Concurrent cs: Preparing students for a multicore world,” in Proceedings of the 13th annual conference on Innovation and technology in computer science education (ITiCSE, ed.), ITiCSE, 2008.

[6] T. W. Burger, “Intel multi-core processors: Quick reference guide,” tech. rep., Intel Corporation, 2005.

[7] L. De Giusti, E. Luque, F. Chichizola, M. Naiouf, and A. De Giusti, “Automatic mapping tasks to cores. evaluating amtha algorithm in multicore architectures.,” in Congreso Argentino de Ciencias de la Computación, 2009.

[8] V. Pankratius, C. Schaefer, A. Jannesari, and W. F. Tichy, “Software engineering for multicore systems: an experience report,” in IWMSE ’08: Proceedings of the 1st international workshop on Multicore software engineering, (New York, NY, USA), pp. 53–60, ACM, 2008.

[9] T. Murphy, “High-performance computing in high schools?,” IEEE Distributed Systems Online, vol. 8, no. 8, pp. 1–3, 2007.

[10] S. Lathrop and T. Murphy, “High-performance computing education,” Computing in Science and Engineering, vol. 10, no. 5, pp. 9–11, 2008. Introducción a la revista escrita por los editores.

[11] J. L. Balcázar, Programación Metódica. McGraw-Hill, 1993.

[12] A. Marowka, “Think parallel: Teaching parallel programming today,” IEEE Distributed Systems Online, vol. 9, pp. 1–8, 2008.

[13] B. Rague, “Teaching parallel thinking to the next generation of programmers,” Journal of Education, Informatics and Cybernetics, vol. 1, no. 1, pp. 43–48, 2009.

[14] S. Carr, J. Mayo, and C.-k. Shene, “Threadmentor: a pedagogical tool for multithreaded programming,” ACM Journal of Educational Resources, vol. 3, pp. 1–30, march 2003.

[15] H. Farian, K. M. Anne, and M. Haas, “Teaching high-performance com-puting in the undergraduate college cs curriculum,” Journal of Computing Sciences in Colleges, vol. 23, no. 3, pp. 135–142, 2008.

[16] C. W. Kessler, “Teaching parallel programming early,” in Proceedings of Workshop on Developing Computer Science Education: How Can It Be Done?, p. 6, March 2006.

[17] A. Grama, A. Gupta, G. Karypis, and V. Kumar, Introduction to Parallel Computing – Second Edition. Pearson Education and Addison Wesley, 2003.

[18] A. De Giusti, Algoritmos, datos y programas con aplicaciones en Pascal, Delphi y Visual Da Vinci. Pearson Education and Prentice Hall, 1 ed., 2001.

[19] R. Champredonde and A. De Giusti, "Design and implementation of the visual da vinci language." Tesina de grado Facultad de Informática UNLP, 1997.

[20] A.-C. J. C. T. Force, "Computer science curriculum 2008: An interim revision of cs 2001," tech. rep., ACM Press, dec 2008.

[21] A. De Giusti and F. Tinetti, "Arquitecturas multiprocesador distribuidas. modelos, software de base y aplicaciones." Proyecto acreditado, 2010.