Una Experiencia de Programación Distribuida utilizando Paradise

Marcelo Mejía Olvera
Instituto Tecnológico Autónomo de México
Río Hondo No. 1, Tizapán San Ángel, 01000 México, D.F.
e-mail: marcelo@lamport.rhon.itam.mx

Resumen

Paradise es un lenguaje paralelo de alto nivel que permite desarrollar fácil y rápidamente aplicaciones distribuidas. La evaluación del desempeño de un sistema real que implementa el algoritmo de visualización *Ray Tracing* en paralelo muestra que es posible obtener factores de aceleración altos y una eficiencia cercana al 100% utilizando el ambiente de programación Paradise.

Palabras clave: Lenguajes de programación distribuida, Algoritmos de visualización.

1 Introducción

Las gráficas por computadora son una de las áreas de la Computación que requieren realizar mayor cantidad de operaciones matemáticas. Una aplicación real de cálculo intensivo en esta área surge en el proyecto "Reconstrucción Tridimensional de Núcleos Celulares de Cromatina" de la Facultad de Ciencias de la UNAM, en el que se reconstruyen computacionalmente, y se despliegan para llevar a cabo un análisis visual certero, imágenes en tres dimensiones provenientes de la digitalización de fotografías de cortes ultrafinos de núcleos de cromatina. La parte correspondiente a la reconstrucción de la imagen sobre un monitor de dos dimensiones representa un grave problema ya que los algoritmos de visualización eficaces necesitan mucho tiempo para ejecutarse.

En este artículo se describen los resultados obtenidos al utilizar una arquitectura paralela que permite incrementar la velocidad de ejecución de un algoritmo de reconstrucción mediante la distribución de la carga de procesamiento entre un conjunto de estaciones de trabajo conectadas en red. En la sección 2 del artículo se introduce brevemente el algoritmo de reconstrucción utilizado y en la sección 3 se describe la implementación paralela realizada del algoritmo utilizando el lenguaje Paradise. En la sección 4 se presentan las medidas de desempeño global del sistema de visualización, mientras que en la sección 5 se presentan las conclusiones del trabajo.

2. Algoritmo de reconstrucción

Para reconstruir una imagen sobre una pantalla de computadora a partir de los cortes digitalizados se optó por implementar en paralelo el algoritmo *Ray Tracing*, que es uno de los más eficientes para

desplegar imágenes tridimensionales en dispositivos bidimensionales de manera realista. El funcionamiento del algoritmo de *Ray Tracing* está basado en principios de óptica geométrica y considera la interacción de rayos de luz con la imagen tridimensional o escena. El algoritmo básico lanza rayos de luz desde el punto de vista que se desea reconstruir hacia adentro de la escena a través de cada uno de los pixeles de una pantalla virtual; estos rayos viajan en dirección contraria a la propagación de los rayos de luz en la realidad (Green, 1991). Para cada rayo lanzado se determina cuál de todas las superficies de la escena es la más cercana a la posición del observador y por lo tanto es la superficie visible en el pixel correspondiente. Para obtener imágenes realistas que incluyan efectos de sombra, reflexiones y refracciones causados en la realidad por rebotes de rayos de luz, el algoritmo normalmente se extiende lanzado rayos secundarios a partir de cada intersección de un rayo primario y una superficie. Hacia cada fuente de luz de la escena se dirigen rayos de sombra para determinar la contribución de éstas a la iluminación del punto de intersección. Dependiendo de las características de las superficies también pueden lanzarse rayos de reflexión o refracción que, si intersectan con otras superficies, pueden también contribuir a la iluminación del punto de origen de estos rayos. Para calcular la iluminación final de un pixel se consideran las intensidades de todos los rayos creados de manera recursiva que influyen en el punto correspondiente de la superficie visible.

Dado que el algoritmo de *Ray Tracing* requiere de un tiempo de procesamiento muy grande, para aumentar su velocidad de ejecución pueden aplicarse restricciones al algoritmo y/o utilizarse técnicas de aceleración. De acuerdo al grado de realismo necesario en la imagen resultante, se decidió que en nuestra implementación del algoritmo podían aplicarse las siguientes restricciones:

- la escena contiene únicamente dos fuentes internas de luz,
- sólo se toman en cuenta dos rebotes para cada rayo primario, y
- se descartan las texturas.

La idea general detrás de las técnicas de aceleración es reducir el número o el costo de las pruebas de intersección de los rayos con las superficies en la escena. En la implementación realizada se utilizó la técnica de voxelización para reducir el número de pruebas de intersección. En esta técnica primeramente se representa la escena como un conjunto de pequeños cubos (voxeles). A continuación, para cada rayo se determina el voxel que contiene la superficie visible y se realizan las pruebas de intersección (sólo) sobre los objetos contenidos en el voxel. Los cálculos de las pruebas de intersección se simplifican en nuestra implementación ya que se considera que los objetos de la escena están formados por esferas pequeñas, que se adaptan a los bordes redondeados de las células. Para disminuir aún más el número de pruebas de intersección la imagen se preprocesa antes de iniciar el algoritmo de *Ray Tracing* para determinar qué elementos constitutivos de cada voxel contienen una esfera y cuáles están vacíos.

En resumen podemos decir que el algoritmo de *Ray Tracing* implantado recibe la información de la escena tridimensional como entrada, la procesa tomando en cuenta las restricciones y las técnicas de aceleración antes mencionadas, y como salida genera un archivo que indica la iluminación correspondiente a cada pixel del monitor.

3. Programación paralela utilizando Paradise

Para implementar el sistema paralelo de visualización se escogió el lenguaje Paradise (SCA, 1994) que representa la tercera generación de un conjunto de lenguajes de programación paralela derivados de Linda (Carriero and Gelernter, 1990). El funcionamiento de estos lenguajes está basado en el uso de una memoria virtual compartida, denominada espacio de tuplas (ET), que es un espacio de datos disponible a cada proceso de una aplicación distribuida. En el ET se almacenan conjuntos ordenados de valores denominados tuplas. La identificación de las tuplas no se realiza mediante direcciones sino mediante un nombre lógico constituido por un subconjunto de sus valores.

El modelo Linda añade al lenguaje C las operaciones de coordinación entre procesos `out()`, `read()` e `in()` para acceder al ET y permitir la comunicación y sincronización entre procesos. La operación `out(t)` permite agregar la tupla `t` al ET; el proceso que invoca la operación continúa su ejecución inmediatamente. La operación `read(s)` lee una tupla `t` del ET que concuerde con `s` (su nombre lógico sea el mismo) y asigna los parámetros actuales de `t` a los formales de `s`. Si no existe en el ET una tupla `t` que concuerde con `s` entonces el proceso que invoca la operación se suspende hasta que se encuentre una concordancia. La operación `in(s)` es similar a la operación `read(s)`, pero retira del ET la tupla `t` que concuerde con `s`. Existen también, en el modelo Linda, las operaciones `inp(s)` y `readp(s)` que son versiones no bloqueantes de `in(s)` y `read(s)`. En Paradise se cuenta, además, con operaciones para crear, abrir, cerrar y destruir espacios de tuplas persistentes.

Paradise es particularmente útil en el contexto de programas distribuidos organizados como una colección de trabajadores idénticos. En este modelo un programa no se particiona en tareas sino que se copia *n* veces, donde *n* está determinado por el número de procesadores disponibles. Un proceso capataz introduce en ET los trabajos a realizar y cada uno de los trabajadores busca continuamente en ET un trabajo para ejecutar. El modelo de trabajadores replicados presenta las ventajas de ser fácilmente escalable en función del número de procesadores disponibles y de no necesitar ninguna interacción directa entre los trabajadores.

En el sistema de visualización paralelo desarrollado se divide la pantalla de la computadora donde se despliegan los resultados en 1600 segmentos rectangulares de 12x12 pixeles cada uno. Se utilizan dos ET en la aplicación. El primero contiene en cada instante el número del siguiente segmento que debe ser procesado. Cada trabajador accede repetidamente al ET_1 para leer (`in`) y actualizar (`out`) este número, y procesa el segmento correspondiente utilizando el algoritmo de *Ray Tracing* para calcular la iluminación de cada uno de sus 144 pixeles (Mejía et al, 1994). Los trabajadores almacenan los resultados del procesamiento de cada segmento en el segundo ET.

El sistema de visualización se ejecuta en un conjunto de estaciones de trabajo RS6000 de IBM y consta de tres etapas principales:

- Preprocesamiento de la imagen
- *Ray Tracing* paralelo
- Desplegado

El funcionamiento del sistema es controlado desde una computadora personal que se encarga inicialmente de preprocesar la imagen, copiar (utilizando rcp) la imagen preprocesada a las estaciones de trabajo del sistema y lanzar (mediante rsh) la ejecución en paralelo de los procesos Paradise que ejecutan el algoritmo paralelo de *Ray Tracing*. Posteriormente, la computadora personal colecta los resultados del algoritmo de reconstrucción que indican la iluminación que debe tener cada uno de los pixeles de la pantalla y despliega la imagen sobre su monitor.

Para maximizar el paralelismo real del sistema, en cada computadora que participa en el algoritmo de *Ray Tracing* paralelo se ejecuta solamente un trabajador. El número de segmentos que procesa cada trabajador es variable, ya que depende de la cantidad de cálculos que necesite el algoritmo de *Ray Tracing* secuencial para determinar la iluminación de los pixeles que le son asignados.

4. Evaluación del desempeño del sistema

Las pruebas realizadas para evaluar el desempeño del sistema paralelo se efectuaron utilizando una imagen típica que nos proporcionó la Facultad de Ciencias de la UNAM y se efectuaron principalmente para contestar dos preguntas: 1) ¿es posible reducir a menos de una hora el tiempo necesario para reconstruir una imagen típica completa (con el realismo deseado) utilizando el sistema paralelo?, y 2) ¿la eficiencia del sistema desarrollado se mantiene alta conforme se aumenta el número de procesadores? La respuesta a la segunda pregunta es crucial ya que si la eficiencia del sistema no se degrada al incrementar el número de trabajadores, entonces puede reducirse el tiempo de procesamiento simplemente aumentando el número de estaciones de trabajo que conforman el sistema.

Antes de presentar los resultados experimentales obtenidos al medir el desempeño del sistema paralelo, definamos las siguientes variables:

- n número de trabajadores que ejecutan el algoritmo,
- Tn tiempo de ejecución del algoritmo usando n trabajadores,
- $s = T1/Tn$ aceleración (relativa) del sistema
- $e = Tn/n$ eficiencia del sistema paralelo

En un sistema paralelo se espera que Tn disminuya al aumentar n y se busca que la aceleración s obtenida sea una función lineal de n, y por lo tanto que la eficiencia e esté cerca del 100%. Las figuras 1 y 2 muestran la aceleración relativa y la eficiencia del sistema paralelo en función de n para subconjuntos de la pantalla de 16 y 160 segmentos, respectivamente. El valor de n está limitado a 9 por el número de licencias disponibles de Paradise

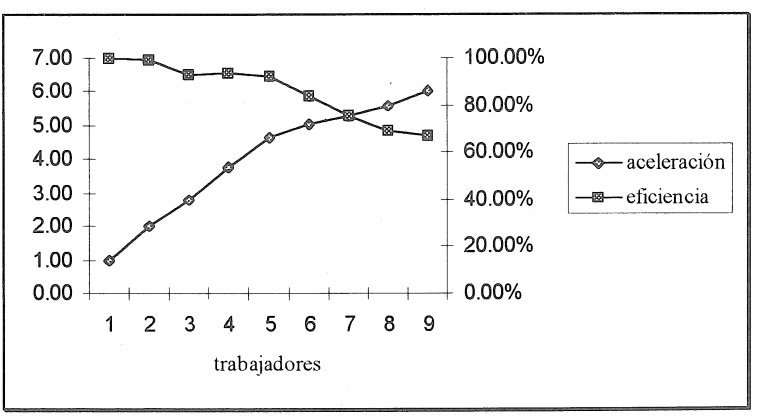

Figura 1. Aceleración y eficiencia para 16 segmentos.

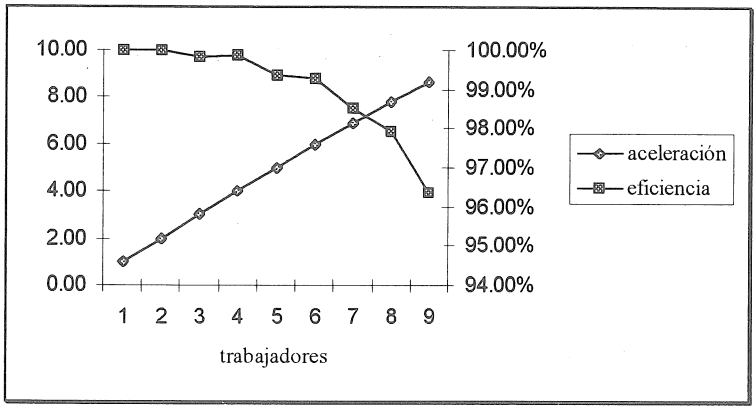

Figura 2. Aceleración y eficiencia para 160 segmentos.

Debe notarse que *T1* es un poco mayor que el tiempo necesario para correr el algoritmo de *Ray Tracing* de manera secuencial (*Ts*) ya que el trabajador consume tiempo al acceder tres veces a los espacios de tuplas (para obtener y actualizar el número del siguiente segmento a procesar, y para escribir el resultado del procesamiento) por cada segmento que debe procesar. Por lo tanto, la aceleración absoluta (*Ts/Tn*) es menor que la aceleración relativa mostrada en las figuras 1 y 2. Por ejemplo, para 16 segmentos y dos trabajadores la aceleración disminuye de 1.99 a 1.76 (*Ts* = 265s, *T1* = 299s y *T2* = 150s). De cualquier forma, la implementación paralela con dos trabajadores es más rápida que el algoritmo secuencial, y la velocidad absoluta aumenta al agregar más trabajadores.

En las figuras 1 y 2 puede observarse que al aumentar el número de segmentos de 16 a 160, la eficiencia del sistema se degrada con mayor lentitud. Esto se debe a que al aumentar el tamaño del

problema, disminuye la proporción del tiempo total del algoritmo paralelo durante el cual existen computadoras sin trabajo al agotarse los segmentos a procesar. Podemos afirmar entonces que si se escogen adecuadamente la complejidad del algoritmo de *Ray Tracing*, la granularidad de la segmentación y el número de trabajadores, el sistema de visualización paralelo es capaz de obtener valores importantes de aceleración y eficiencia.

Utilizando el máximo de 9 trabajadores, el sistema paralelo procesó los 1600 segmentos correspondientes a una pantalla completa en 2894s (menos de 49 minutos). Dado que $T1$ = 2987s para 160 segmentos y $T9$ = 2894s para 1600 segmentos, podemos afirmar que el desempeño de la implementación paralela se escala bien con el tamaño del problema y el número de trabajadores dentro de los límites de nuestras pruebas.

6. Conclusiones

El uso del paralelismo en computación es de gran utilidad en aplicaciones que requieren grandes cantidades de cálculos matemáticos. En el campo de la graficación tradicionalmente se han utilizado supercomputadoras vectoriales, como la CRAY YMP, o máquinas masivamente paralelas, como la CM-5. El trabajo presentado en este artículo muestra que es posible utilizar estaciones de trabajo sencillas conectadas en red para implantar algoritmos, como el *Ray Tracing*, que consumen mucho tiempo de procesador. En este sentido se observa la eficacia de la programación paralela en una ambiente distribuido.

El algoritmo de *Ray Tracing* puede paralelizarse de distintas maneras. En la implantación realizada se consideró un paralelismo de grano burdo que se adapta bien al ambiente de ejecución distribuido. La repartición de la carga de trabajo en términos de grupos de pixeles que constituyen segmentos de la imagen, y la posibilidad de variar el tamaño de los segmentos en función de la complejidad de la escena, permiten obtener una buena relación entre el tiempo que cada computadora dedica al procesamiento de la imagen y el tiempo que emplea para comunicarse (indirectamente a través del ET) con otros procesos.

El uso de un lenguaje de programación paralelo de alto nivel como Paradise facilita el trabajo de desarrollo de aplicaciones distribuidas y disminuye el tiempo de diseño, implementación y pruebas comparado con el uso de llamadas de bajo nivel al sistema como son los sockets.

En Paradise no existe la operación `eval("nombre",f())` de Linda que permite crear procesos dinámicamente. En el sistema de visualización paralelo los procesos son creados por medio de un proceso externo a Paradise. Para automatizar la creación de procesos y realizarla al interior de un proceso Paradise, se han creado en el ITAM recientemente librerías que hacen posible ejecutar procesos Paradise en máquinas específicas; estas librerías se encargan de copiar el código ejecutable en la máquina remota y de lanzar la ejecución del proceso.

Referencias

Carriero, N., and Gelernter, D. (1990); How to Write Parallel Programs - A First Course; The MIT Press.

Green, S. (1991); Parallel Processing for Computer Graphics; The MIT Press.

Mejía, M., Hernández, H., y Quiroz, J. (1994); Visualización Tridimensional en Paralelo; Congreso Nacional de Informática, Fundación Arturo Rosenblueth, México, D.F.

SCA (1994); Paradise - User's Guide and Reference Manual; Scientific Computing Associates.