ANDES: Un Analizador de Desempeño para Programas Paralelos

Oscar Naim y Alejandro Teruel
Departamento de Computación y Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080A, Venezuela
e-mail: emsca!usb!teruel@SUN.COM

Resumen: Se presenta una herramienta automatizada que obtiene mediciones para realizar análisis del desempeño de programas paralelos en máquinas MIMD con memoria distribuida. Esta herramienta, denominada *ANDES* (ANalizador de DESempeño), es un *monitor de software* que inserta código adicional para recoger y almacenar información resumida relativa a los *eventos* que definen el desempeño del programa.

La herramienta se apoya en una serie de *métricas* para analizar el desempeño de programas paralelos (aceleramiento, eficiencia, fracción serial empíricamente determinada, porcentaje de tiempo ocioso por procesador, balance de carga, balance de comunicación, tiempo de sincronización y porcentaje de solapamiento de comunicación con cálculo). Se propone una metodología simple de análisis de desempeño que puede aplicarse conjuntamente con ANDES y se presentan brevemente algunos de los resultados de las experiencias vividas con la herramienta.

El prototipo de ANDES fue desarrollado para analizar programas escritos en el lenguaje ANSI C Paralelo de INMOS, sobre un anillo bidireccional de transputadores.

Palabras claves: Paralelismo, desempeño, rendimiento, métricas, C, transputadores.

1. Antecedentes y motivación del trabajo

Los métodos y herramientas convencionales para analizar desempeño en programas secuenciales, no son adecuados en un ambiente distribuido. El esfuerzo que tiene que hacer un programador para realizar el análisis del desempeño de un programa paralelo sin ayuda automatizada es muy grande. El programa a analizar puede contener miles de líneas de código y cientos de rutinas, transformándose en una tarea tediosa y propensa a errores la inserción de nuevo código en el programa, para generar la información relacionada con el desempeño, así como también la de realizar el análisis correspondiente.

Se han reportado herramientas experimentales para programas paralelos con metas similares, tales como [2], [3], [4], [5], [9], [12], [13], [14] y [16], diferenciándose entre ellas por el conjunto de métricas que aplican, el grado de interacción, la calidad de la interfaz visual, tipo de grafos utilizados, nivel de apoyo de hardware y el tipo de máquina paralela a que están orientadas.

2. Descripción y limitaciones de ANDES

ANDES, es un *monitor de software* que analiza el desempeño de un programa insertando código adicional que permite recoger y almacenar información resumida relativa a los eventos que definen el desempeño del programa. Esta información, sin embargo, está dispersa a lo largo de toda la red de procesadores y es local a cada procesador. Una vez que la aplicación termina su ejecución, ANDES recoge y almacena los resultados obtenidos y le ofrece al usuario la posibilidad de analizarlos.

A continuación, se describen las tres partes principales que constituyen ANDES:

a.- *Configurador*

Le permite al usario introducir al sistema los datos particulares de su configuración de hardware, tales como topología y canales de comunicación. Para este trabajo y para los efectos del prototipo, la topología será la de un anillo bidireccional. De esta manera, el configurador se reduce a un archivo de configuración, donde se especifican los canales de entrada y salida de cada uno de los procesadores en el anillo.

b.- *Preprocesador*

Una vez especificados los datos relacionados a la configuración del sistema, el preprocesador se encarga de insertar el código necesario en el programa para recolectar eventos que permitan determinar el comportamiento del desempeño del programa. Estos datos, se resumen en cada procesador hasta que el programa culmine su ejecución, transmitiéndose luego al procesador central. Todas las ocurrencias de eventos en ANDES son *registradas* y *resumidas* en un *árbol dinámico de llamadas* de la aplicación. La rutina principal de la aplicación es la raíz del árbol. Luego, cada vez que se invoca a una *instancia* de una rutina, se crea un nuevo nodo del árbol que enlaza al padre (rutina que invoca) con el hijo (rutina invocada). Cada nodo del árbol, lleva un registro de la información asociada a esa rutina, es decir, el tiempo total de corrida, el tiempo de comunicación, el tiempo de no comunicación, etc. Una vez finalizada la aplicación, este árbol se almacena en disco para luego ser postprocesado.

c.- *Visualizador*

Provee facilidades gráficas para visualizar los resultados obtenidos. La versión del prototipo de ANDES que se presentará en este trabajo no incluye el módulo *Visualizador*. Trabajos futuros extenderán el prototipo para incluirlo.

ANDES posee una serie de limitaciones entre las cuales se encuentran:

- Todos los programas paralelos analizados por ANDES deben ser *finitos*

- ANDES está pensado para analizar aplicaciones que tienen un tiempo de ejecución relativamente grande, es decir, que consumen minutos u horas ejecutándose.

- Como ANDES es un monitor de software introduce cierto grado de *invasividad* en el programa.

- El usuario debe especificar las características particulares de la topología sobre la cual correrá ANDES a través del configurador. En el caso del prototipo, esta topología está restringida a un anillo bidireccional.

- El prototipo de ANDES sólo analiza programas escritos en lenguaje C Paralelo de INMOS [6] y que están contenidos en un sólo archivo.

3. Qué medir y cómo hacer mediciones en sistemas paralelos

Los objetivos principales que se persiguen al realizar mediciones en sistemas paralelos son:

- Conocer el desempeño del programa:
  Le permite al programador conocer si su aplicación está ejecutándose satisfactoriamente o no. De esta manera, el programador determina si vale la pena hacer un análisis detallado del comportamiento de la ejecución de la aplicación, para averiguar las posibles razones por las cuales no se obtiene el rendimiento esperado.

- Hacer proyecciones y determinar escalabilidad:
  Una vez que se conoce el desempeño de la aplicación, es importante plantearse las siguientes preguntas: ¿cómo se comportaría la aplicación si se ejecutara sobre una plataforma con un número diferente de procesadores? Las respuestas a estas preguntas determinan hasta dónde es escalable la aplicación, es decir, hasta qué número de procesadores es factible crecer sin influir negativamente en el desempeño de la misma.

- Ubicar cuellos de botella:
  Si el desempeño de la aplicación no es el esperado, es necesario ubicar aquellas partes del programa dónde se encuentran los problemas que originan la disminución del rendimiento. Una vez que se identifican estos cuellos de botella, se determina dónde concentrar esfuerzos de entonación.

- Obtener información suficiente para conocer e incrementar el grado de paralelismo del programa:
  Ciertas arquitecturas, como los transputadores, permiten realizar comunicación y cálculo de manera simultánea en un mismo procesador, aprovechando al máximo el paralelismo de una aplicación. Sin embargo, ¿se está solapando al máximo el total de tiempo posible en el que se puede hacer comunicación con cálculo? La respuesta puede ser clave en el proceso de entonación de la aplicación.

Con esto presente, describiremos algunas de las *métricas* más útiles: aceleramiento (*speedup*), eficiencia y fracción serial empíricamente determinada [8]. El resto de las métricas que usa ANDES son porcentaje de solapamiento de comunicación con cálculo, porcentaje de tiempo ocioso por procesador, balance de carga, balance de comunicación y tiempo de sincronización y se describen en [10].

El aceleramiento, es el tiempo de corrida del mejor algoritmo secuencial, dividido por el tiempo de corrida de la versión paralela del algoritmo requerido por ρ procesadores.

La eficiencia se define como el cociente entre el aceleramiento y el número de procesadores. Una eficiencia pequeña, indica que se están desperdiciando recursos.

Es obvio que agregando más procesadores se debería reducir el tiempo de corrida, pero ¿en qué medida?. El aceleramiento se encarga de medirlo. Aceleramiento cercano a lineal es bueno, pero ¿qué significa suficientemente cercano a lineal?. Aquí es donde entra en juego la fracción serial. La fracción serial es una medida de la tasa de cambio de la eficiencia y, como tal, permite estimar la proyección del aceleramiento para un número determinado de procesadores.

La fracción serial *teórica* de un programa, es la parte del mismo que debe ejecutarse obligatoriamente de manera secuencial. Calcular de una manera analítica esta fracción serial es, en general, un problema no trivial.

Por esta razón, Karp introdujo en [8] la definición de *fracción serial empíricamente determinada* (fracción serial de aquí en adelante), la cual es una aproximación práctica de la fracción serial teórica. La fracción serial se define de la siguiente manera:

$$f = \frac{\frac{1}{A(n,p)} - \frac{1}{p}}{1 - \frac{1}{p}} \qquad (1)$$

siendo A(n,p) el aceleramiento.

Una ventaja muy importante de la fracción serial, es que permite realizar proyecciones acerca del comportamiento del programa sobre un número mayor de procesadores, determinando la *escalabilidad* del algoritmo.

La fracción serial es una buena herramienta de diagnóstico para detectar posibles problemas en el desempeño de un programa, debido a que mientras el aceleramiento y la eficiencia varían a medida que el número de procesadores crece, la fracción serial podría permanecer constante en un sistema ideal. Variaciones en la fracción serial constituyen valiosas pistas en el análisis de desempeño.

4. Metodología para analizar desempeño

La siguiente metodología tiene como objetivo principal, analizar el desempeño de un programa para determinar si es susceptible a mejoras o no, es decir, aplicarle *performance debugging*. Los pasos de la metodología propuesta son los siguientes:

Paso 1:

Objetivo Responder las siguientes preguntas: ¿qué se tiene?, ¿dónde estamos ubicados?

Métricas usadas: Aceleramiento, eficiencia.

Descripción

El aceleramiento y la eficiencia, permiten determinar si el desempeño es aceptable. Sin embargo, si el aceleramiento no es aceptable, no son suficientes para discernir la causas que originan este comportamiento a pesar que situaciones como las siguientes constituyen *advertencias posibles* de problemas de desempeño:

- El aceleramiento no crece linealmente con el número de procesadores.
- El aceleramiento crece linealmente hasta cierto número de procesadores y, a partir de allí, comienza a disminuir hasta llegar a deteriorar las mejoras de tiempo anteriores.
- El número de puntos de comparación (e.g. 2 y 4 procesadores) es muy pequeño como para poder estimar con precisión si la paralelización del algoritmo ha sido realmente exitosa. Mientras mayor sea el número de puntos de comparación utilizados, nuestras conclusiones estarán más y mejor sustentadas, de lo contrario podríamos llegar incluso a conclusiones erróneas.

Paso 2:

Objetivo Realizar proyecciones y estimar cuánto se puede mejorar.

Métricas usadas: Fracción serial, Ley de Amdahl

Descripción

- Calcular la fracción serial para p=1,..,P (donde P es el número total de procesadores) y obtener su línea de proyección, es decir, cómo se comporta a medida que aumenta el número de procesadores.
- Aplicar la ley de Amdahl, usando la fracción serial apropiadamente, para determinar el máximo aceleramiento posible para un número de procesadores dado.

Es posible que sea difícil determinar si el aceleramiento de una aplicación es lo suficientemente bueno (por ejemplo, un aceleramiento de 70 sobre 100 procesadores ¿es suficiente o no?). Para ello, se utilizan la fracción serial y la ley de Amdahl. A través de estas métricas se puede determinar si, aumentando el número de procesadores, la parte *paralela* del programa crece más rápidamente que la *serial* incrementándose por lo tanto el aceleramiento.

Paso 3:

Objetivo ¿Dónde concentrar esfuerzos? (Análisis por rutina).

Métricas usadas: Balance de carga, balance de comunicación, porcentaje de tiempo de comunicación y no comunicación, porcentaje de solapamiento de comunicación con cálculo, porcentaje de tiempo consumido en sincronizar procesos.

Descripción

- Ubicar las rutinas más *pesadas* de la aplicación, es decir, aquellas rutinas que consuman mayor cantidad de tiempo para saber dónde concentrar esfuerzos.
- Determinar cuellos de botella específicos en esas rutinas.

Paso 4:

Objetivo Aplicar mejoras y repetir 1,2 y 3 para validar los resultados.

Descripción

Una vez que se han identificado los problemas y se le han aplicado los correctivos necesarios a la aplicación, el proceso de evaluación vuelve a comenzar sobre la aplicación ya mejorada. Este proceso de evaluación y mejora iterativo continúa hasta llegar a un resultado satisfactorio con respecto al comportamiento del desempeño de la aplicación.

5. Experiencias con ANDES

Para evaluar ANDES con un problema real y además probar las bondades de la metodología de análisis propuesta, Acosta y Fernández [1] implementaron una versión paralela del algoritmo *Gibbs Sampler* y la analizaron usando ANDES.

El algoritmo del *Gibbs Sampler* tiene como propósito general, obtener una curva que represente la función asociada a una serie de datos iniciales utilizando para ello métodos estadísticos.

Este algoritmo posee un grado de paralelización muy alto, debido en primer la lugar, a la posibilidad de repartir el trabajo a realizar por cada procesador en forma equitativa y, en segundo lugar, a que no existe comunicación entre los procesadores a lo largo de la ejecución del algoritmo y solamente se transmiten los resultados finales.

Los resultados obtenidos por Acosta y Fernández utilizando mediciones directas, fueron los mismos que los obtenidos por ANDES. El aceleramiento, se ubicó entre 3.72 y 3.87 para 4 procesadores (no se disponían de más procesadores para realizar pruebas).

El aceleramiento más bajo, 3.72, fue generado precisamente por uno de los problemas más representativos. Al analizar la fracción serial, esta resultó ser del 2%. Sin embargo, el porcentaje de tiempo ocupado por procesador fue del 96% en promedio y el balance de carga se ubicó en 0.96![0] Estos resultados indican que el balance de carga es muy bueno y pensando que no existe comunicación a lo largo del algoritmo pareciera contradictorio obtener una fracción serial del 2%. Una posible explicación, la cual originó una discusión muy interesante, es que el algoritmo del *Gibbs Sampler* tiene una duración *aleatoria* dependiendo del punto en la secuencia de datos iniciales en que comience. Además, la secuencia

[0] Este resultado se obtiene al dividir el valor promedio entre el máximo, indicando en qué porcentaje nos estamos alejando de la media (el valor ideal es 1).

de instrucciones que ejecuta el algoritmo puede variar dependiendo precisamente de los resultados intermedios obtenidos. Esto quiere decir que el problema que se ejecutó sobre un procesador no es exactamente el mismo que el que se ejecutó sobre 4 procesadores, siendo este último en nuestro caso más *pesado* que el primero, causando esta situación un efecto leve sobre la fracción serial.

En estas y otras pruebas preliminares de ANDES, el overhead causado por las mediciones del prototipo de ANDES llegó a ser del 14% en la ejecución de procesos de más de un minuto). Sin embargo, el overhead para la medición del tiempo de comunicación fue de apenas 2%.

Los resultados de otras experiencias, incluyendo un algoritmo paralelo para la descomposición LU de matrices, son descritos en [10].

6. Conclusiones

- ANDES es un monitor de software que analiza el comportamiento del desempeño de programas paralelos con el objeto de aplicarles *performance debugging*: es decir, mejorar el desempeño del programa a través de un proceso de entonación iterativo. ANDES está basado en las siguientes métricas: aceleramiento, eficiencia, fracción serial, balance de carga, tiempo total de corrida, tiempo de comunicación y no comunicación, tiempo de sincronización entre procesos, balance de comunicación, porcentaje de solapamiento de comunicación con cálculo, porcentaje de tiempo ocioso por procesador
- El prototipo de ANDES es una herramienta útil para realizar análisis de desempeño de programas paralelos, como lo demuestran experiencias como el desarrollo del *Gibbs Sampler* paralelo.
- Con respecto al uso de las métricas propuestas en [10], podemos decir que la incorporación del algoritmo de Jones descrito en [7] para estimar el tiempo en cual un procesador está "ocupado", es decir, realizando labores de cálculo, ha dado resultados muy útiles al ser utilizada junto con la fracción serial empíricamente determinada. También la medición del tiempo de solapamiento de comunicación con cálculo entre dos procesos que se están ejecutando en paralelo en el mismo procesador ha resultado particularmente útil.
- El prototipo de ANDES resultó ser una herramienta muy fácil de usar. En SIMPAR[11], por ejemplo, el proceso de medición se llevó a cabo en tres días, incluyendo el análisis de alrededor de doscientas páginas de números! Sin embargo, las mediciones hechas utilizando ANDES se efectuaron en menos de media hora para el análisis del algoritmo *Gibbs Sampler* descrito en el capítulo V. Aún tomando en cuenta la diferencia de magnitud entre los dos programas (el prototipo de SIMPAR es unas 10 veces más grandes que el *Gibbs Sampler*) la mejora es significativa.
- El overhead causado por las mediciones del prototipo de ANDES llegó a ser del 14%.

7. Recomendaciones y Direcciones Futuras

En base a las experiencias antes descritas hacemos las siguientes recomendaciones:

- Desarrollar el módulo de *visualización*
- Desarrollar un módulo *configurador* que permita extender el prototipo a otras topologías e incluso tratar de hacerlo independiente de la topología.
- Hacer el prototipo más portable e intentar migrarlo a otros ambientes (e.g. transputadores T9000).
- Comparar el prototipo con otras herramientas tanto comerciales como experimentales.
- Determinar el grado de invasividad del prototipo.
- Buscar mecanismos de software, hardware o híbridos más precisos para estimar el tiempo de sincronización entre procesos. Los resultados obtenidos para el prototipo indican que la precisión de la medición es deficiente.

8. REFERENCIAS

[1] D. Acosta, M. Fernández *Generación de Variables Aleatorias en Paralelo* Trabajo de grado de la carrera de Ingeniería de la Computación, Universidad Simón Bolívar. Enero, 1992.

[2] H. Burkhart, R. Millen "Performance-Measurement Tools in a Multiprocessor Environment". *IEEE Transactions on Computers*; Vol. 38, No. 5, May 1989.

[3] A. Couch, D. Krumme *Monitoring Parallel Executions in Real Time* IEEE Computer Society, 1990.

[4] E. Gabber "VMMP: A Practical Tool for the Development of Portable and Efficient Programs for Multiprocessors". *IEEE Transactions on Parallel and Distributed Systems*; Vol. 1, No. 3, July 1990.

[5] W. Harrison "Tools for Multiple-CPU Environments". *IEEE Software*; May 1990.

[6] INMOS. *Ansi C Toolset User Manual* INMOS Limited, 1990.

[7] G. Jones et al. *Measuring the Busyness of a Transputer* Occam User Group Newsletter. No. 12, January 1990.

[8] A. Karp, H. Flatt "Measuring Parallel Processor Performance". *Communications of the ACM*; May 1990, Vol. 33, No. 5.

[9] R. McLaren, W. Rogers *Instrumentation and Performance Monitoring of Distributed Systems* IEEE Computer Society Press, 1990.

[10] O. Naim *Un Analizador de Desempeño para Programas en C Paralelo (ANDES)*. Tesis de Maestría, Universidad Simón Bolívar, Caracas, Venezuela, Enero 1992.

[11] O. Naim, A. Teruel *Consideraciones sobre el Paralelismo en SIMPAR (FASE I)* Reporte IT-1991-001, Depto. de Computación y Tecnología de la Información. Universidad Simón Bolívar, Caracas, Venezuela, 1991.

[12] K. Nichols "Performance Tools". *IEEE Software*; May 1990.

[13] D. Pease, A. Ghafoor, I. Ahmad, D. Andrews, K. Foudil-Bey, T. Karpinski, M. Mikki, M. Zerrouki "PAWS: A Performance Evaluation Tool for Parallel Computing Systems". *Computer*; January 1991.

[14] M. Reilly *A Performance Monitor for Parallel Programs* Academic Press Inc., 1990.

[15] M. Sanabria, A. Hung *Construcción de un Parser de ANSI C* Trabajo de grado de la carrera de Ingeniería de la Computación, Universidad Simón Bolívar, Caracas, Venezuela. Octubre, 1991.

[16] A. Wagner, S. Chanson, N. Goldstein, J. Jiang, H. Larsen, H. Sreekantaswamy *TIPS: Transputer -based Interactive Parallelizing System* Department of Computer Science, University of Columbia, 1991.