Desarrollo de un cluster computacional para la compilación de algoritmos en paralelo en el Observatorio Astronómico.

John Jairo Parra Pérez[♠]

Resumen

Este artículo muestra cómo funciona la supercomputación y además como esta puede contribuir al desarrollo de futuras investigaciones en varios campos del conocimiento; tomando para ello los recursos computacionales que brinda la Universidad y software que se encuentra libre en Internet. La razón por la cual se está desarrollando este proyecto es que sirva de herramienta para las futuras investigaciones a realizar en el Observatorio Astronómico.

Abstract

The super computation theory actually this is the principal tool to develop investigations in much areas of knowledge. In Astronomy this is very important in the galactic dynamics, cosmology and others areas. By this reason the Astronomical Observatory of Sergio Arboleda University is making a computational cluster for futures investigations.

[♠] Ingeniero de Sistemas de la Universidad Autónoma de Colombia, procesamiento de datos y operación del telescopio del Observatorio Astronómico de la Universidad Sergio Arboleda.

Palabras Clave: Cluster, kernel, paralelizar, speedup, OpenMosix.

1. Introducción.

La supercomputación ha contribuido a la solución de problemas en muchas áreas del conocimiento tales como la astronomía, ingeniería, física, matemáticas, paleontología, criminalística, medicina forense, solo por citar algunas; incluso en nuestro entretenimiento como por ejemplo en los espectaculares efectos digitales casi imperceptibles que caracterizan las producciones de la pantalla gigante. Nace de la necesidad del ahorro de grandes sumas de dinero que implicaría invertir en sistemas computaciones muy veloces. Un cluster trabaja bajo el procesamiento en paralelo y se define como la capacidad de utilizar varios procesadores para ejecutar diferentes partes del mismo programa simultáneamente[1]. El objetivo principal del paralelismo es el reducir el número de ciclos de ejecución de un programa en relación al número de procesadores que existen en el sistema.

Para que el software compile en el cluster necesita ser paralelizado, es decir hacer que su código permita la distribución de procesos. Para ello se cuenta con muchas herramientas las cuales son de acceso libre en Internet; pero no todos los programas pueden ser paralelizados así que en la mayoría de los casos es tarea de los grupos de diseño del cluster desarrollar el software que permita solucionar su problema.

[1] The OpenMosix HOWTO.

En astronomía la implementación de un cluster es indispensable sobre todo en las áreas de reducción y análisis de datos y construcción de modelos de diferentes tipos tales como dinámica estelar, dinámica de galaxias, cosmología y muchos mas. Por esta razón, para poder estudiar nuestros campos de interés se dio paso a su realización.

2. Teoría de la supercomputación.

2.1. La Ley de Amdahl

Un programa con un número de instrucciones determinado tardará un tiempo *t* en ejecutarse de manera completa en un sistema uniprocesador. La medida de mejora de rendimiento (o *speedup*) toma como referencia el tiempo de ejecución de un programa en un sistema uniprocesador respecto a la medida de tiempo de ejecución del programa en un sistema multiprocesador o multicomputador.

$$speedup = \frac{t_1}{t_j}$$

Donde t_1 es el tiempo que tarda en ejecutarse el programa para una máquina con un solo procesador y t_j es el tiempo que tarda en ejecutarse el mismo programa con un sistema de *j* procesadores. el *speedup* o mejora de rendimiento aumenta linealmente con el número de procesadores que posea el sistema. Para el caso de los programas lo primero que debe hacerse es comprender su naturaleza, luego clasificar su código: cuál es paralelizable y cuál es lineal ya que este último en varias ocasiones va a requerir tener

acceso a ciertas dependencias. Para ello debe existir un nodo en el cual se ejecuten este código y del que depende el resto. Es lógico afirmar que el *speedup* de un programa depende de:

a) El tiempo en el que se ejecuta el código lineal W_l, llamado así por trabajo lineal.

b) El tiempo en el que se ejecuta el código paralelizable W_n.

c) El numero de procesadores o nodos que posee el sistema n.

Dados estos factores se puede concluir que el máximo *speedup* que se puede obtener en un programa a nivel general viene dado por:

$$speedup = \frac{W_l + W_n}{W_l + \frac{W_n}{n}}$$

Esta es la llamada Ley de Amdahl y fue descrita por Gene Amdahl en 1967. Las implicaciones que trae esta ecuación son claras a pesar de que no se tenga en cuenta las características de cada sistema en concreto, que a su vez también limitarían el *speedup* de cada sistema. Para empezar el rendimiento no depende completamente del número de procesadores que posea el sistema: en la mayoría de los casos dependería del número de procesadores máximos que se aprovecharía para ejecutar un programa. Por otro lado puede concluirse que cuanto mejor paralelizado esté un programa mías susceptible será de aumentar su *speedup* y por tanto explotar el rendimiento de un sistema paralelo como puede ser un cluster.

Por ejemplo si tenemos un programa que inicialmente no se ha paralelizado, y si en nuestras medidas el tiempo de ejecución lineal es del 12% y el de paralela es 88% *(Figura 1)*

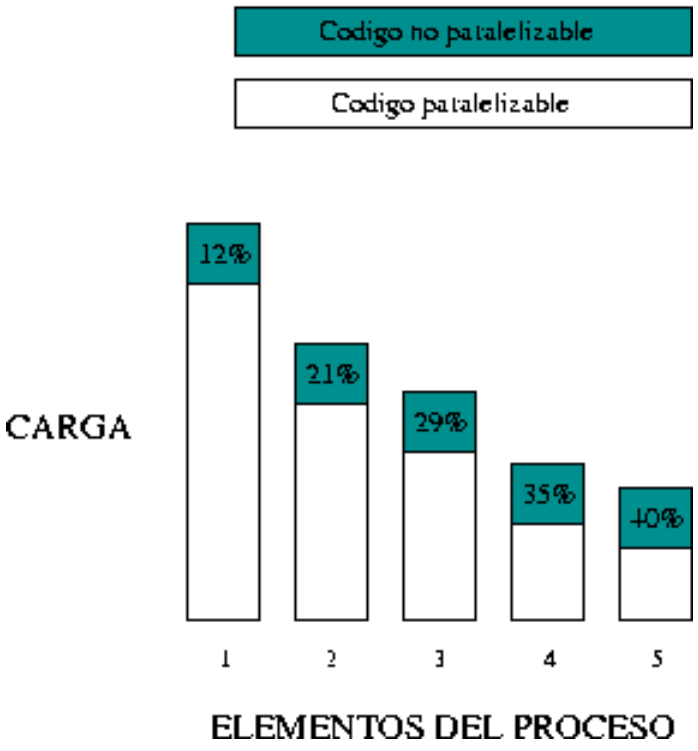

Figura 1: Ejemplo del incremento del speedup obtenido por la Ley de Amdahl.

En la figura la parte no paralelizable del proceso impide que se pueda escalar de forma lineal, llegará un momento en que añadir nuevos procesadores no añada una ventaja real al sistema porque casi todo lo que esté haciendo cada uno de los procesadores sea procesar código secuencial. Por lo tanto para maximizar el aprovechamiento de los sistemas paralelos se debe tener mucho cuidado con la forma de paralelizar las aplicaciones; ya que entre más código secuencial tengan, más problemas de escalabilidad aparecerán.

2.2. Hardware.

En lo que se refiere a límites del hardware, la eficiencia en la paralelización es lo mejor que se posee, ya que se paraleliza por instrucción en cada procesador de la computadora. Respecto a este tema existe gran variedad de mejoras de rendimiento a los límites que se imponen por el programa, estas mejoras se van incorporando en los procesadores de última generación. El problema es que esta solución puede resultar en muchos casos demasiado costosa, si no es por la adquisición de la computadora, por la dificultad o tiempos de implementación. Además siempre se necesita cierta ayuda de parte del software, principalmente el compilador. Así aunque el PentiumIV tenga nuevas instrucciones vectoriales, no son prácticamente usadas por los compiladores actuales y el rendimiento de un PIV es bastante inferior al esperado. En cambio usando el compilador especialmente desarrollado por Intel, los mismos programas aprovechan las instrucciones vectoriales del procesador mucho mejor con lo que se consiguen programas mucho más rápidos. Hoy en día es difícil ver programar en ensamblador[2] para procesadores que no estén dedicados a tareas como dispositivos empotrados o de tiempo real, e incluso en estos este lenguaje se hace más raro cada día, a no ser que sea software de sistema. Por lo tanto las optimizaciones de uso de las nuevas capacidades de los procesadores tienen que venir de parte del compilador que es quien se encarga de trabajar a bajo nivel.

[2] El ensamblador es un lenguaje de programación de bajo nivel.

2.3. Software.

Por lo visto anteriormente, la Ley de Amdahl pone restricciones al incremento de rendimiento en un sistema que use código no paralelizable. La única solución posible es la optimización del código así que en el ciclo de vida del software se deberá tener en cuenta la paralelización. Además existe otro tipo de limitaciones comenzando por el sistema elegido para paralelizar aplicaciones. Ahora bien el papel del compilador es sumamente importante, este se clasifica en: compilador tonto y compilador listo (*Figura 2*).

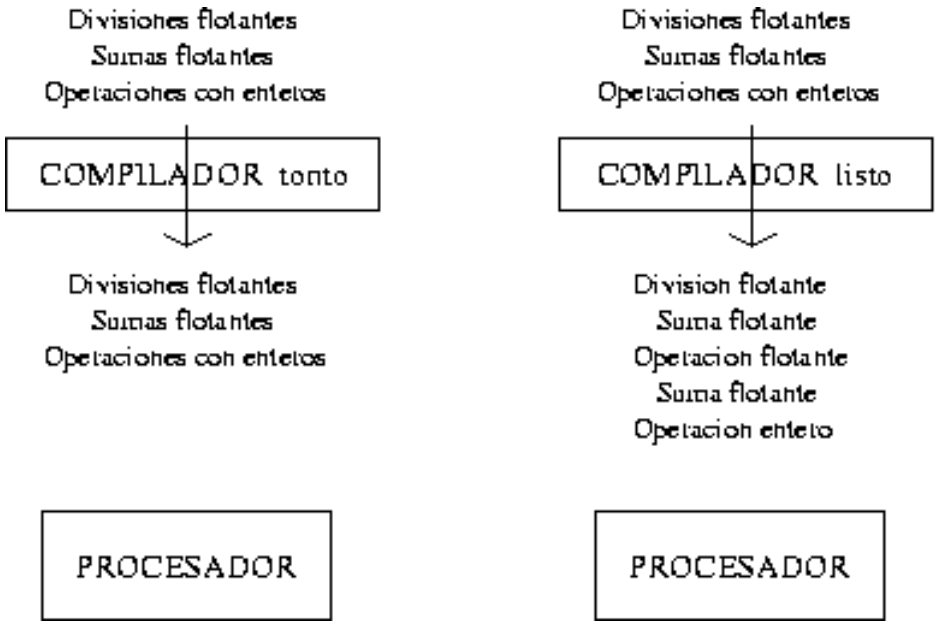

Figura 2: La función del compilador.

Dicho papel puede verse en acción entre un compilador inteligente frente a uno que no lo es tanto ante una misma entrada: un programa con una secuencia de divisiones con coma flotante, después unas sumas del mismo tipo y por último unas operaciones con

enteros, todas sin dependencias entre sí y ocurre que: el compilador tonto no las reordena con lo que si el procesador no tiene ordenación de instrucciones hará que las instrucciones de enteros que seguramente tardan 1 o 2 ciclos tengan que esperar a las instrucciones de división en coma flotante, operaciones que pueden tardar más de 30 ciclos. Mientras se hacen las divisiones en coma flotante no se están usando las unidades de suma de flotantes ni las de operaciones con enteros.

En cambio el compilador listo primero ejecuta la instrucción que más tiempo tardará en ejecutarse, y después ejecuta las instrucciones que usan otras unidades para que el procesador pueda estar procesando de forma paralela todos los datos, usando al máximo su hardware. Por supuesto que esta misma operación (primero las operaciones que más tardan) si se realiza sobre las mismas unidades hace que el retardo medio de finalizar una instrucción aumente por lo que el compilador debe hacer las elecciones correctas si está intentando realizar una ordenación inteligente.

3. Construcción del Cluster.

En el observatorio contamos con siete computadoras disponibles para la elaboración del cluster de las cuales cuatro poseen las mismas especificaciones de hardware, a estas se les instaló Linux como sistema operativo y el cluster se está trabajando con el software OpenMosix. Por ahora se está configurando cada uno de los nodos para que trabaje óptimamente y se deben realizar pruebas con código en paralelo.

4. Conclusiones.

Durante el trabajo que se ha realizado con OpenMosix se han determinado que entre las ventajas que posee, está el no requerir paquetes extra para su operación, además no es necesario realizar modificaciones de códigos; pero como desventaja se tiene la dependencia directa con el *kernel* y la presencia de problemas con la memoria compartida.

Por las pruebas hasta ahora efectuadas, para una operación óptima del cluster todos los equipos deben tener las mismas especificaciones técnicas.

Bibliografía.

[1] K. Buytaert: *The OpenMosix Howto*, v0.3, 2003.

[2] M. Colomer: *Clustering with OpenMosix,* 2003.

[3] M. Bar: *HPC Computing Applied to Business Applications,* 1997.

[4] I. Latter: *Heterogeneous Clusters,* v1.1, 2003.

[5] J. Ridruejo, J. Agirre, J. M. Alonso: *Estrategias de instalación y gestión de clusters con software libre,* 2003