## UN POCO DE HISTORIA

El benchmark Linpack fue desarrollado en el Argone National Laboratory por Jack Dongarra en 1976, y es uno de los más usados en sistemas científicos y de ingeniería.

Su uso como benchmark fue accidental, ya que originalmente fue una extensión del programa Linpack -cuyo propósito era resolver sistemas de ecuaciones- que otorgaba el tiempo de ejecución del programa en 23 máquinas distintas. Luego fueron agregándose cada vez mayor cantidad de máquinas (según sus mismos autores más como un pasatiempo que otra cosa).

El benchmark Linpack puede conseguirse en versión Fortran, C y como un applet de Java.

## Descripción del benchmark (WIKI)

La característica principal de Linpack es que hace un uso muy intensivo de las operaciones de coma flotante, por lo que sus resultados son muy dependientes de la capacidad de la FPU que tenga el sistema. Además pasan la mayor parte del tiempo ejecutando unas rutinas llamadas BLAS (*Basic Linear Algebra Subroutines* o Subrutinas de Álgebra Lineal Básica). Como hay dos tipos de estas bibliotecas (una codificada en ensamblador y otra en Fortran), el resultado también dependerá mucho de esto. De lejos, el mayor tiempo de ejecución se consume en la rutina DAXPY de la biblioteca BLAS (casi el 90%). DAXPY realiza el siguiente cálculo

y(i) := y(i) + a * x(i).

Es por esto que en realidad se puede decir que lo que mide Linpack es la velocidad del sistema para DAXPY.

Por otra parte, al realizar esencialmente cálculos con matrices es un test fácilmente paralelizable, y se puede utilizar para medir la eficiencia de sistemas multiprocesador (de hecho, existe una página en Internet que informa el "Top 500" de las computadoras basándose en el Linpack).

DESCRIPCIÓN 2 (NETLIB.ORG)

**HPL** es un paquete de software que soluciona (al azar) un sistema denso lineal en doble precisión (64 bits) aritmética en los equipos de memoria distribuida. Por lo tanto, puede ser considerado como portable.

El **algoritmo** utilizado por HPL se puede resumir en las siguientes palabras clave: bidimensional de distribución de datos en bloque cíclico.

El paquete HPL proporciona un programa de pruebas y tiempo para cuantificar la **exactitud** de la solución obtenida, así como el tiempo que se tardó en calcularla. El mejor **rendimiento** alcanzable por este software en su sistema depende de una gran variedad de factores. Sin embargo, con algunos supuestos restrictivos sobre la red de interconexión, el algoritmo descrito aquí y su aplicación adjunta son **escalable** en el sentido de que su eficiencia se mantiene constante en paralelo con respecto al uso por memoria del procesador.

## 3 benchmarks en 1

Linpack está compuesto de tres benchmarks: los ya mencionados de orden 100 y orden 1.000, y un tercero de Computación Altamente Paralela (un algoritmo especialmente diseñado para buscar los beneficios de los multiprocesadores).

En el primero sólo se permite cambiar las opciones del compilador para hacerlo más eficiente; sin embargo debe especificarse antes de correrlo una función SECOND que devuelva el tiempo de ejecución en segundos del programa.

Las reglas básicas para el segundo benchmark son un poco más relajadas. Se puede especificar cualquier ecuación lineal que uno desee, implementada en el lenguaje que uno desee.

## Optimización

El Linpack es un programa que ejecuta cálculos con matrices, y como tal, existe numerosas técnicas para optimizarlo. Por ejemplo, intentando hacer un uso más eficiente de la carga de datos en la caché más próxima, puede llevar a una aproximación de cálculos matriz-vector o matriz-matriz. Las técnicas a aplicar dependerán de los tamaños de los distintos niveles de caché y la arquitectura de la computadora. A continuación detallamos otra técnica:

## Loop unrolling (desarrollo del bucle)

El loop unrolling es una técnica de optimización que consiste en hacer más extenso un bucle simple reduciendo la sobrecarga por comparación del bucle y permitiendo una mejor concurrencia. De esta manera, la siguiente estructura repetitiva:

```
PARA i:= 1 a n HACER
    y(i) := y(i) + alfa * x(i)
FIN PARA
```

En lugar de hacer una sola instrucción por bucle, se pueden lograr 4 instrucciones por bucle (o las que uno desee) de la siguiente forma:

```
m := n - n MOD 4
PARA i:= 1 a m, PASO 4 HACER
    y(i) := y(i) + alfa * x(i)
    y(i+1) := y(i+1) + alfa * x(i+1)
    y(i+2) := y(i+2) + alfa * x(i+2)
    y(i+3) := y(i+3) + alfa * x(i+3)
FIN PARA
PARA i: m a n HACER
    y(i) := y(i) + alfa * x(i)
FIN PARA
```

## RESULTADOS

Los resultados se expresan en MFlops (millones de operaciones de coma flotante por segundo), siempre referidas a operaciones de suma y multiplicación de doble precisión (64 bits, a unque para algunos sistemas esta cantidad de bits es la precisión simple).

```
Input data or print help ? Type [data]/help :

Number of equations to solve (problem size): 20000
Leading dimension of array: 20000
Number of trials to run: 4
Data alignment value (in Kbytes): 4
Current date/time: Sun Dec 14 19:00:27 2008

CPU frequency:    3.400 GHz
Number of CPUs: 4
Number of threads: 4

Parameters are set to:

Number of tests                             : 1
Number of equations to solve (problem size) : 20000
Leading dimension of array                  : 20000
Number of trials to run                     : 4
Data alignment value (in Kbytes)            : 4

Maximum memory requested that can be used = 3200404096, at the size = 20000

============= Timing linear equation system solver =================

Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm)
20000  20000  4      111.673    47.7658  4.546476e-10 4.024627e-02
20000  20000  4      111.306    47.9233  4.546476e-10 4.024627e-02
20000  20000  4      111.339    47.9091  4.546476e-10 4.024627e-02
20000  20000  4      111.733    47.7400  4.546476e-10 4.024627e-02

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
20000  20000  4       47.8345  47.9233

End of tests
```