Apéndice B: Rendimiento de Procesamiento Secuencial de las Computadoras

Este Apéndice se dedica a la descripción del rendimiento de las computadoras que se obtiene en el campo de aplicación definido de los problemas de álgebra lineal, específicamente los que se pueden expresar en función de multiplicaciones de matrices.

La caracterización del rendimiento de las computadoras es un problema muy conocido y estudiado por varias razones, esencialmente para la estimación del tipo de problemas que se pueden resolver en términos de tamaño y tiempo de ejecución.

Si bien es importante la caracterización del rendimiento de las computadoras, también se describe en este Apéndice cómo afecta al rendimiento la optimización del código que se lleva a cabo. Además, se muestra cómo esta optimización tiene efectos variables dependiendo de las características del hardware de cada computadora.

En el caso específico de aplicaciones paralelas, conocer el rendimiento secuencial con precisión es esencial para caracterizar con esa misma precisión la ganancia obtenida por el aporte del procesamiento paralelo. De la misma manera, se puede estimar o al menos se tiene la información mínima necesaria para estimar la relación del costo-beneficio de las computadoras paralelas en general y de las redes de computadoras procesando en paralelo en particular.

B.1 Introducción

La caracterización del rendimiento de las computadoras se ha utilizado con varios propósitos, entre los cuales se pueden mencionar [7]:

- Estimación de la capacidad de resolución de problemas, tanto en lo referente al *tamaño* de los problemas que se pueden resolver como al *tiempo de ejecución* necesarios.
- Verificación del costo de las computadoras, no solamente en cuanto al hardware sino también en cuanto al software de base y de aplicación necesarios.
- Elección de la computadora más adecuada para el problema o clase de problemas que se deben resolver. Implícitamente en este caso se utiliza el índice de rendimiento como un parámetro de comparación de las computadoras a utilizar.

Tradicionalmente, la capacidad de cómputo numérico de una computadora se ha caracterizado con la cantidad de operaciones de punto flotante por unidad de tiempo (Mflop/s: millones de operaciones de punto flotante por segundo) o por un número que lo identifique de forma unívoca (SPEC: [6] [15]). Tradicionalmente también se han tomado dos líneas generales para el cálculo de este índice de rendimiento:

1. Análisis del hardware de procesamiento: unidad/es de punto flotante, diseño de las unidades de punto flotante (pipelines, registros internos, etc.), memoria/s cache (niveles, tamaños, etc.), capacidad de memoria principal, etc.
2. Ejecución de un programa o conjunto de programas específicos de cálculo denominados *benchmarks*.

El análisis del hardware de procesamiento normalmente da lugar a lo que se conoce como *rendimiento pico*, o rendimiento máximo *teórico* de la computadora. Esta línea de caracterización del rendimiento ha sido adoptada normalmente por los fabricantes de las computadoras y también ya es aceptado que es muy poco probable de obtener por una aplicación específica.

La utilización de benchmarks se hizo cotidiana dada la separación existente entre el rendimiento pico y el rendimiento real que las aplicaciones obtienen normalmente en su ejecución en las computadoras. Es muy difícil la elección de un conjunto de programas que logren reunir las características de, o representar a, *toda* la gama de posibles programas que se pueden ejecutar sobre una computadora y por lo tanto existen muchos benchmarks utilizados y aún muchos más propuestos.

Si está bien definido el tipo de aplicaciones específicas sobre el cual se utilizarán las computadoras, sigue siendo muy útil la caracterización en este campo de aplicaciones sin utilizar los benchmarks más generales. Este es el caso de las aplicaciones definidas en términos de las multiplicaciones de matrices [2] [4], y por lo tanto lo más preciso que se puede obtener en este campo es el rendimiento de la multiplicación de matrices misma, que se tomará como el *benchmark* de referencia para la experimentación.

Utilizar un *benchmark* tan específico y tan cercano a la aplicación que se debe resolver tiene, en el contexto de la aplicación paralela, una ventaja más: define con precisión la velocidad relativa de las computadoras para el procesamiento local. Si bien este índice (velocidad relativa de cómputo) no es tan necesario ni importante en el contexto de las

computadoras paralelas con elementos de procesamiento homogéneos, se torna indispensable para el cómputo paralelo con elementos de procesamiento heterogéneos. Sin este tipo de información es muy difícil llegar a tener balance equilibrado de la carga computacional.

B.2 Computadoras Utilizadas

Elegir computadoras para caracterizar rendimiento secuencial e intentar representar con esta caracterización *todos* los casos posibles es casi tan difícil como elegir el conjunto de programas que permitan comparar el rendimiento de las computadoras. En el caso del cómputo paralelo sobre redes locales ya instaladas, está claro que se deben analizar computadoras interconectadas en una red local, pero la variedad de posibilidades es muy grande.

Las tres redes locales sobre las cuales se trabajará son las que se describieron en detalle en el Apéndice A:

- CeTAD: Centro de Técnicas Analógico-Digitales, Departamento de Electrotecnia, Facultad de Ingeniería, Universidad Nacional de La Plata. Es la que está instalada desde hace más tiempo y las computadoras que la componen son utilizadas con múltiples propósitos.
- LQT: Laboratorio de Química Teórica, CEQUINOR, Departamento de Química, Facultad de Ciencias Exactas, Universidad Nacional de La Plata. Es una red destinada a la resolución de problemas numéricos, fue instalada hace varios años y se ejecutan trabajos secuenciales y paralelos desarrollados con PVM y Linda.
- LIDI: perteneciente al Laboratorio de Investigación y Desarrollo en Informática, Facultad de Informática, Universidad Nacional de La Plata. Está dedicada a enseñanza de programación paralela e investigación. Puede considerarse directamente una instalación más del tipo Beowulf, aunque no de las más costosas en cuanto a cantidad de máquinas y red de interconexión.

Tanto la red del CeTAD como la del LQT tienen varias características interesantes para el estudio de rendimiento con el objetivo de utilizarlas para cómputo paralelo:

- Preexistencia: no se construyeron para ser estudiadas sino para ser utilizadas. En este sentido son *reales* y han tenido la evolución en el tiempo que suelen tener las redes locales con la actualización e inclusión de computadoras.
- Heterogeneidad: se puede considerar como una consecuencia de su preexistencia al presente trabajo de investigación, pero es importante remarcar que ambas redes de computadoras tienen máquinas bastante diferentes entre sí al menos en términos de velocidad de cómputo, y en el caso de la red del CeTAD incluso a nivel de arquitectura de las computadoras.
- Dedicadas a cómputo secuencial: si bien en ambas redes locales se ejecutan programas paralelos, al menos en su instalación las computadoras estaban dedicadas a cómputo secuencial. De hecho, varias de las computadoras de la red local del CeTAD existen para tareas estrictamente secuenciales y fueron adaptadas para cómputo paralelo instalando el software necesario.

- Hardware estándar y de bajo costo: tanto las computadoras como las redes de interconexión son ampliamente conocidas y de bajo costo. De hecho, más de una computadora utilizada puede considerarse lista para ser dada de baja.

La red del LIDI provee el marco estándar de estudio de procesamiento paralelo en redes de computadoras. Más específicamente: el hardware de cómputo y de comunicaciones es homogéneo y la red de interconexión puede considerarse apropiada dado que coincide con las ideas básicas de una instalación Beowulf.

En el Apéndice A se describen las tres redes locales con todas las características consideradas importantes de cada una de las computadoras. Por otro lado, es importante (y de alguna manera es uno de los objetivos más importantes de este Apéndice), identificar las características generales de rendimiento en las redes de computadoras más allá de los detalles propios de cada máquina utilizada. Por lo tanto, se intentará llegar a conclusiones más generales de rendimiento sobre hardware de cómputo heterogéneo.

B.3 Descripción de los Experimentos

Se llevaron a cabo tres grandes grupos de experimentos, dependiendo de la forma de codificar-diseñar-implementar la solución al problema:

1. Código sin optimizar. La multiplicación de matrices se codifica con las tres iteraciones clásicas, sin ningún esfuerzo por parte de la implementación ni por el compilador.
2. Código optimizado por el compilador. En este caso, se compila el código sin optimizar (las tres iteraciones) con todas las opciones de optimización disponibles del compilador utilizado para el procesador de la computadora.
3. Código optimizado a nivel del programa fuente. En este caso, el esfuerzo de optimización del código es realizado a nivel del código fuente (sin recurrir a código de máquina), además opcionalmente se pueden utilizar las opciones de optimización del compilador pero no se considera necesario.

Para cada tipo de optimización se calcula el rendimiento obtenido para varios tamaños de matrices posibles. La variación del tamaño del problema a resolver (en requerimientos de memoria y cómputo) es usual en este tipo de *benchmarks* y tiene por objetivo la identificación de posibles dependencias del rendimiento sostenido de una computadora con respecto al tamaño de la aplicación.

En las subsecciones que siguen se describen:

- Las características más importantes junto con el/los objetivos de cada uno de los tipos de optimizaciones. Siempre en el contexto de procesamiento secuencial y de los problemas numéricos en general y de procesamiento de matrices en particular.
- Los tamaños de matrices elegidos para estimar el rendimiento de las computadoras, y los criterios por los cuales se eligen.
- Las características más importantes de la experimentación en cada computadora, que de alguna manera es una visión de mayor nivel de abstracción para los dos puntos anteriores.

B.3.1 Código sin Optimización

El rendimiento obtenido con la multiplicación de matrices sin optimizar en realidad no tiene gran utilidad como índice de rendimiento ya que sin ningún esfuerzo por parte del programador se puede obtener algo mejor utilizando correctamente las opciones de compilación disponibles. En este caso es necesario solamente conocer bien el compilador y es muy útil conocer las características del procesador.

El rendimiento obtenido con el código sin optimizar se utilizará con dos objetivos:
1. Como referencia de ganancia en cada computadora para los otros dos casos de optimización: del compilador y del código fuente.
2. Como referencia para comparación de los efectos de optimización en computadoras heterogéneas.

B.3.2 Optimizaciones del Compilador

En el caso de las optimizaciones del compilador, no se tienen grandes desventajas a nivel conceptual. Como se ha explicado, se debe conocer el compilador y las características del procesador relacionadas con las operaciones aritméticas y/o diseño de la/s memoria/s cache. Sin embargo, siempre es conveniente tener en cuenta que usualmente el mejor compilador para cada computadora es el que provee el fabricante. De acuerdo a la evolución en el desarrollo y comercialización de las computadoras, inicialmente el compilador era provisto por la empresa fabricante sin costo adicional. Desde hace algunos años, el compilador es optativo y debe comprarse la licencia al fabricante (del código ejecutable y las librerías propias) por separado. Esta situación, tiene al menos dos consecuencias en la actualidad:
1. Muchas de las computadoras instaladas no tienen el compilador (más apropiado) propietario de la empresa fabricante de la computadora.
2. Se hizo más popular aún la utilización de compiladores de uso libre tales como gcc/gcc-egcs para el lenguaje C, que se pueden obtener vía Internet en código binario y/o fuente.

Desde el principio se ha establecido que se pretenden utilizar las computadoras tal como están instaladas, y en todo caso con el mínimo costo de instalación de software (en particular, compiladores) adicional. Por lo tanto, no solamente es recomendable utilizar las computadoras con un compilador de uso libre como gcc/gcc-egcs sino que además se debería cuantificar su eficiencia en cada máquina. El hecho de caracterizar el rendimiento de las computadoras con un compilador en particular de alguna manera también está caracterizando el rendimiento del compilador mismo. Más allá de las conclusiones particulares a las que se llegue respecto del compilador, y que en principio no son materia de estudio en esta tesis, se llega a que:
- En general, el compilador que instalado en cada máquina será el utilizado. Se asume que es lo *mejor* que puede haber o por lo menos lo que tiene mínimo costo.
- La combinación computadora-compilador utilizados puede no ser la mejor y por lo tanto es útil para caracterizar la computadora en particular, pero no para comparar máquinas

en general (a nivel de modelos de computadoras, por ejemplo).

De acuerdo con esta última conclusión se adopta en todos los gráficos de caracterización de rendimiento (en este Apéndice y en todos los de experimentación) la norma de hacer referencia a las computadoras por sus nombres (*hostnames*), sin referencia a marcas y/o modelos. Si bien los nombres de las máquinas no tienen ninguna información (*a priori*) relacionada con el rendimiento y eso puede oscurecer un poco la interpretación de resultados, no es correcto caracterizar rendimiento a nivel de marcas-modelos de computadoras cuando el compilador utilizado no es el *mejor* (asumiendo que el *mejor* es el provisto por el fabricante de la máquina) que se puede utilizar en cada computadora. Por otro lado, en el Apéndice A se dan *todos* los detalles de cada una de las computadoras y en la interpretación de los resultados se hará mención explícita a cada uno de los que se consideren importantes.

B.3.3 Optimización del Código Fuente

Es muy difícil asegurar *a priori* que la optimización del código fuente que se haga es efectiva. Sin embargo, en el campo del cómputo numérico en general existe mucha experiencia (y publicaciones) respecto de las formas de aprovechar al máximo las características del hardware de procesamiento. Algunas de las optimizaciones que ya se consideran estándares a nivel de código fuente son [3] [1] [10] [11] [5]:

- Utilización intensiva de los registros internos dedicados a datos numéricos.
- Procesamiento por bloques para el aprovechamiento intensivo de los datos asignados en memoria/s cache/s.
- Aprovechamiento máximo de las unidades de ejecución implementadas con *pipelines* (identificación de los saltos y dependencias).
- Intercalación de instrucciones que operan con datos de punto flotante y con datos enteros, para el máximo aprovechamiento posible de los procesadores superescalares.
- Identificación y *separación* de las operaciones con dependencias de datos que reducen el rendimiento de los procesadores superescalares.

En el caso particular de las multiplicaciones de matrices, ya se tienen disponibles numerosas fuentes de optimización, y numerosas fuentes de información, inclusive a nivel de código fuente disponible en Internet. Por un lado, se puede llegar a esperar que cada procesador tenga provisto su propio conjunto de rutinas de cómputo numérico, como el caso particular del Pentium III y Pentium 4 de Intel, que disponen de numerosas rutinas en código fuente (aunque en lenguaje de ensamblador, a nivel de instrucciones del procesador) [9] disponibles, que se pueden obtener vía Internet [13].

Si bien las bibliotecas optimizadas por las propias empresas fabricantes de procesadores pueden ser muy atractivas (y de uso libre), de todas maneras subsisten al menos dos inconvenientes:

- No todos los procesadores tienen disponible tal tipo de código-bibliotecas.
- El código fuente suele ser bastante dependiente de un compilador en particular, en el caso anterior de Intel se necesitan compiladores propietarios (cuya utilización depende de comprar una licencia) tanto del lenguaje C como de lenguaje de ensamblador.

Por otro lado, ya hay disponible también en Internet numerosas librerías que generan código optimizado en general, normalmente en lenguaje C o FORTRAN para ser compiladas localmente en las computadoras independientemente del compilador instalado [3]. Uno de los proyectos más significativos al respecto, y aún en desarrollo es el denominado ATLAS (Automatically Tuned Linear Algebra Software) [12] [14]. Como casi la mayoría de los paquetes en el campo del álgebra lineal comenzó dedicado exclusivamente a la multiplicación (secuencial) de matrices y posteriormente se extendió a todo LAPACK Level 3 [4]. Esta alternativa es satisfactoria, por varias razones:

- Es de uso libre, el *único* costo es el de instalación. Esto normalmente implica instalar código que se obtiene en Internet y que ejecuta un conjunto de programas (*scripts*) para identificar el hardware de procesamiento y optimizar el código de acuerdo a lo estimado por estos programas. Finalmente quedan disponibles un conjunto de bibliotecas que contienen las rutinas optimizadas y que se pueden utilizar desde los programas.
- Los conceptos de optimización son suficientemente claros como para ser implementados independientemente de una biblioteca o distribución en particular. Esto significa que ni siquiera es necesario utilizar una biblioteca disponible en Internet sino que se puede desarrollar el código apropiado con su consiguiente costo.
- Es medianamente independiente del procesador, ya que se puede hacer en código fuente como C o FORTRAN y por lo tanto se puede compilar localmente en cada computadora.

Dado que el código se optimiza sin hacer uso específico de un compilador, las opciones de optimizaciones propias del compilador no suelen mejorar mucho el código ejecutable. Una vez más, es necesario recordar que estas optimizaciones son apropiadas para este tipo de procesamiento matricial y no necesariamente puede hacerse en todos los casos o para todas las aplicaciones en general.

La optimización de código fuente también será denominada *optimización completa*, dado que en general es lo mejor que se puede hacer para obtener rendimiento óptimo o cercano al óptimo de cada computadora.

B.3.4 Tamaños de Matrices a Multiplicar

Los tamaños elegidos de las matrices a multiplicar (se asumen matrices cuadradas, de $n \times n$ elementos) tienen relación con las características del procesamiento de matrices y del subsistema de memoria de cada computadora. A modo de ejemplo: si las matrices son suficientemente pequeñas como para ser asignadas completamente en el primer nivel de memoria cache (L1 Cache), el rendimiento sostenido será muy satisfactorio y con valores cercanos al óptimo teórico del procesador. Si, por el contrario, las matrices no pueden ser asignadas en ninguno de los niveles de memoria cache, el rendimiento dependerá en forma casi directa con el patrón de acceso a los datos para ser procesados.

Dada la heterogeneidad de las computadoras con las que se experimenta (Apéndice A), y en particular los diferentes tamaños y niveles de memoria cache de las computadoras, se tomaron como referencia varios tamaños de matrices relativamente pequeños con respecto al tamaño de memoria principal: matrices de orden n = 100, 200, 400. Los datos numéricos se representan con números en punto flotante de precisión simple (norma IEEE 754 [8]) de

cuatro bytes. Por lo tanto, para n = 100, la cantidad de datos necesaria para almacenar una matriz será de $100^2 \times 4$ bytes, un poco menos de 40 KB de datos.

Dos valores se tomaron como representativos de los tamaños de matrices que se pueden manejar en memoria principal de 32 MB: matrices de orden n = 800 y de orden n = 1600. Con matrices de 800×800 elementos, la cantidad de memoria requerida para contener las tres matrices que intervienen en una multiplicación ($C = A \times B$) es de aproximadamente 7.3 MB (22.8% del total de 32 MB de memoria principal aproximadamente). En el caso de matrices de 1600×1600 elementos, la cantidad aproximada de memoria que se requiere es 29.3 MB, lo que representa el 91.6% del total de 32 MB de memoria principal.

En el caso de las computadoras con 32 MB de memoria principal se puede considerar suficiente el tamaño del problema con n = 1600, o por lo menos es suficientemente grande como para que el tamaño de la memoria cache no sea suficiente para contener una parte relativamente grande del problema. Sin embargo, teniendo en cuenta que hay máquinas con 512 MB de memoria principal, se buscaron valores de n suficientemente grandes para ocupar casi completamente la memoria principal.

Por lo tanto, en todas las computadoras se realizaron los experimentos con matrices cuadradas de orden n = 100, 200, 400, 800 y 1600. En el caso de las computadoras con memoria principal de 64 MB o 512 MB, y para tener valores de referencia para ser utilizados en el cálculo de speedup, se llevaron a cabo experimentos con matrices mayores. Además, dado que siempre se tiende a la utilización de las computadoras en el límite de su capacidad, también se llevaron a cabo experimentos con el máximo posible en cuanto a tamaño de las matrices. Como es de suponer, esto depende no solamente del tamaño de la memoria principal instalada sino también del espacio de swap configurado en el sistema.

Para las computadoras de 64 MB de memoria principal, los tamaños considerados representativos de los problemas que requieren una buena parte o toda la memoria principal corresponden a valores de n = 1900, 2000, 2200 y 2400. Estos tamaños de matrices implican los porcentajes aproximados de requerimientos de memoria (asumiendo 64 MB en total) de: 65%, 72%, 87%, y 103% respectivamente. Se debe recordar que es posible experimentar con los valores cercanos y superiores al 100% de requerimientos de memoria principal dependiendo del tamaño de memoria swap configurada.

En las computadoras de 64 MB de memoria principal, el tamaño máximo con el cual se pudo llevar a cabo la multiplicación de matrices es para n = 3200, y como referencia se hicieron también experimentos con n = 3000. Estos tamaños de matrices implican los porcentajes aproximados de requerimientos de memoria (asumiendo 64 MB en total) de: 183% y 161% respectivamente. Como se puntualizó antes, los tamaños máximos del problema dependen de tres aspectos:

- memoria principal instalada.
- espacio de swap configurado.
- sistema operativo, ya que es éste el que en última instancia decide cuándo cancelar un proceso por falta de memoria.

Y estos tres aspectos coinciden al menos en la máquinas más rápidas de la red del CeTAD y de la red del LIDI.

Para las computadoras de 512 MB de memoria principal, los tamaños considerados representativos de los problemas que requieren una buena parte o toda la memoria principal corresponden a valores de n = 4000, 5000, 6000 y 7000. Estos tamaños de matrices implican los porcentajes aproximados de requerimientos de memoria (asumiendo 512 MB en total) de: 36%, 56%, 80%, y 110% respectivamente. Se debe notar que es posible experimentar con los valores cercanos y superiores al 100% de utilización de memoria principal para almacenar los datos dependiendo del tamaño de memoria swap configurada.

En las computadoras de 512 MB de memoria principal, el tamaño máximo con el cual se pudo llevar a cabo la multiplicación de matrices es para n = 9000, y como referencia se hicieron también experimentos con n = 8000. Estos tamaños de matrices implican los porcentajes aproximados de requerimientos de memoria (asumiendo 512 MB en total) de: 181% y 143% respectivamente.

En resumen, en las computadoras más rápidas de cada red local se experimentó en los límites de la capacidad total de memoria. En todos los casos, cuando se muestra el rendimiento de cada computadoras se muestra también cuál es el máximo tamaño de matrices que se puede resolver sin hacer uso del *swapping* de páginas de memoria.

B.4 Rendimiento de la Multiplicación de Matrices

En las subsecciones que siguen se muestran los tiempos de ejecución y también el rendimiento expresado en Mflop/s obtenidos en cada una de las computadoras que resuelven una multiplicación de matrices según el tipo de optimización realizada. Cuando se considera necesario, se incluyen también algunos comentarios que explican los valores de los índices de rendimiento de las computadoras. Dado que las ocho computadoras de la red local del LIDI son iguales, se muestran todos los datos de la experimentación solamente para una de ellas. Finalmente, se incluye una subsección más dedicada a la comparación del rendimiento entre los distintos tipos de optimización elegidos.

B.4.1 Rendimiento sin Optimización

La Figura B.1 muestra los tiempos de ejecución (registrados en segundos) obtenidos para la multiplicación de matrices (cuadradas) de diferentes tamaños en las computadoras del CeTAD. La Figura B.2, muestra el tiempo de ejecución en las computadoras del LQT. En el eje **x** del gráfico se muestra el valor de n (tamaño de las matrices) y en el eje **y** se muestra el tiempo de ejecución en valores logarítmicos.

Si bien tanto la Figura B.1 como la Figura B.2 permiten tener una idea aproximada de los tiempos de ejecución necesarios en cada una de las máquinas, la escala logarítmica y el mismo índice de rendimiento definido como tiempo de ejecución puede complicar la interpretación de los resultados. De todas maneras, se puede identificar rápidamente que para un tamaño de matrices definido, las diferencias de velocidad de cómputo son notables.

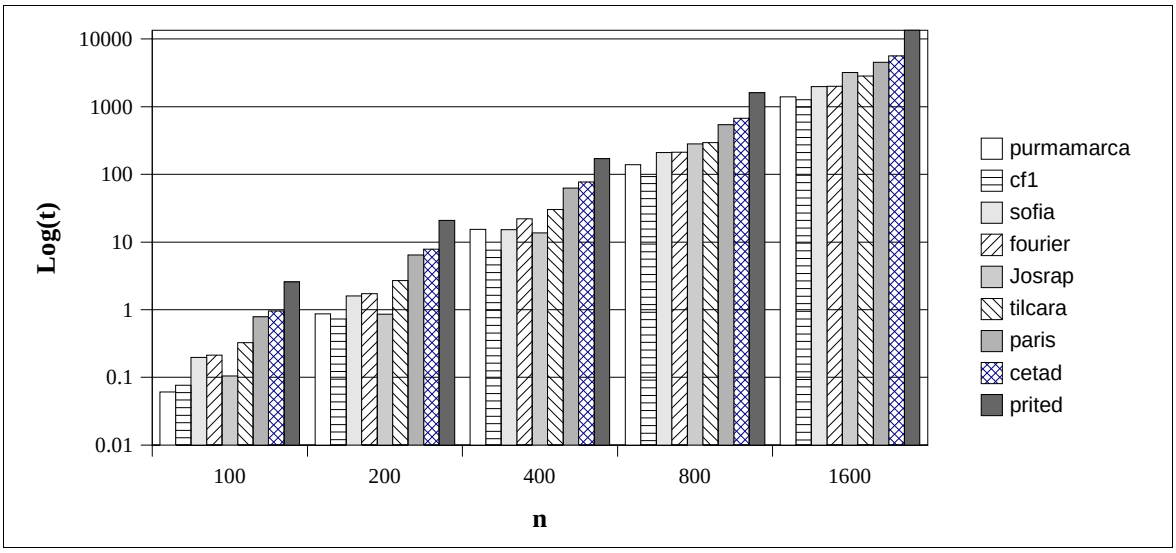

Figura B.1: Tiempos de Ejecución en el CeTAD sin Optimización.

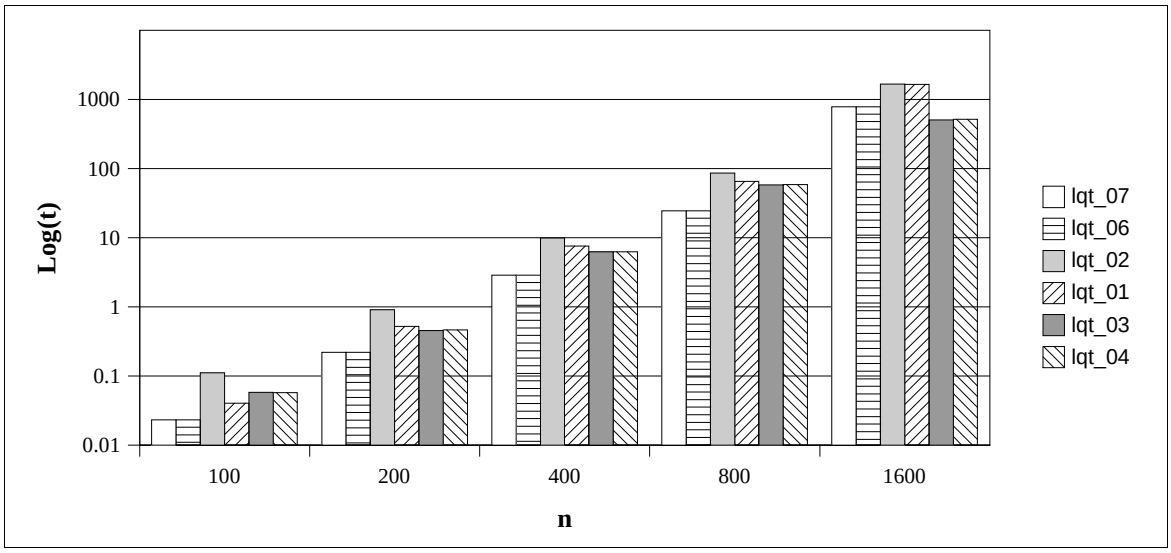

Figura B.2: Tiempos de Ejecución en el LQT sin Optimización.

La Figura B.3 así como la Figura B.4 muestran los mismos experimentos pero identificando los Mflop/s obtenidos en computadora. Lo más notable que se puede identificar con facilidad son las diferencias de velocidad de cómputo de cada máquina que queda bastante enmascarada en el caso de los gráficos de tiempo con escala logarítmica.

En todas las computadoras es notable el efecto que tiene el tamaño de la memoria cache y el tamaño del problema a resolver sobre el rendimiento. En todos los casos, a medida que la cantidad de datos crece, la probabilidad de reusar un dato asignado en la memoria cache disminuye dado que no se establece a priori ningún patrón de acceso a los datos para aprovechar la jerarquía de memoria con uno o más niveles de memoria cache.

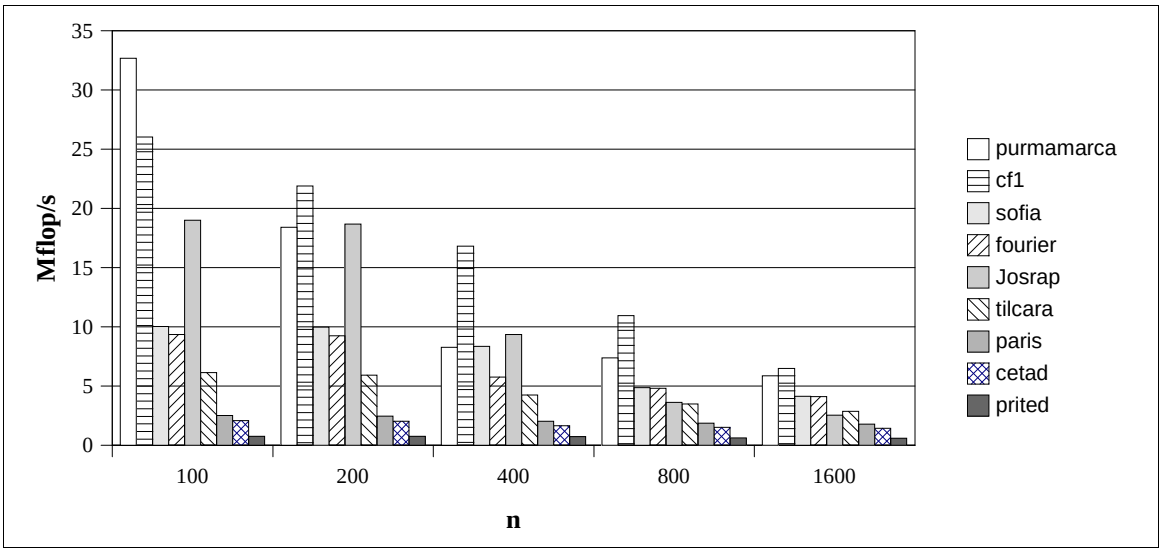

Figura B.3: Mflop/s en el CeTAD sin Optimización.

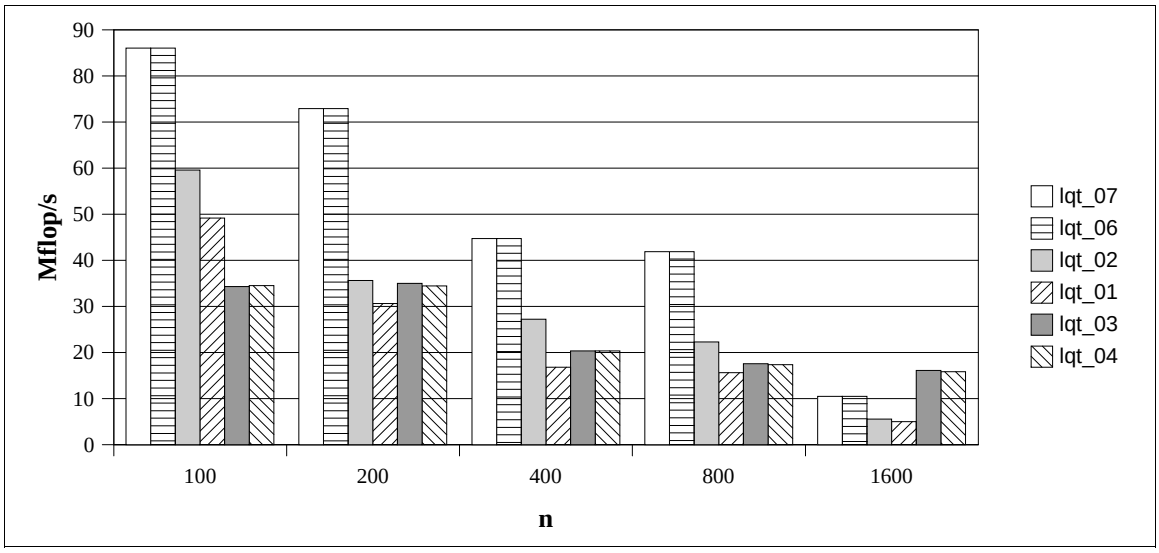

Figura B.4: Mflop/s en el LQT sin Optimización.

Es interesante notar que a medida que los procesadores son más veloces, el impacto sobre la disminución del rendimiento cuando el problema crece es mayor, dado que la diferencia relativa de velocidad de memoria y de procesamiento crece proporcionalmente. Por ejemplo, **lqt_01** pasa de casi 50 Mflop/s con matrices de 100x100 a 5 Mflop/s con matrices de 1600x1600 lo que implica utilizar solamente una décima parte del rendimiento posible tal como fue medido para los problemas con matrices de orden n = 100.

Las diferencias de velocidad relativa entre el acceso a memoria y el procesamiento generan a su vez diferencias relativas de velocidad entre las computadoras que dependen del tamaño del problema a resolver. Por ejemplo, para n = 100, la computadora con denominada **paris** (Figura B.3), tiene casi el doble de la capacidad de cálculo de **Josrap** y para n = 1600 **Josrap** tiene mayor rendimiento que **paris**. Si bien para cómputo secuencial esto da una idea de velocidad relativa diferente para distintos tamaños de problemas,

cuando se trata de cómputo paralelo tiene impacto directo en el balance de carga computacional, que debería ser resuelto por la aplicación en función del tamaño del problema de procesamiento de datos.

B.4.2 Rendimiento con Optimizaciones del Compilador

Como se explicó anteriormente, los programas de aplicación que se ejecutan normalmente tienen el grado de optimización mínimo que provee el compilador para la computadora (más específicamente, para el procesador). En todas las máquinas reportadas, el compilador utilizado es *gcc/gcc-egcs* y por lo tanto las opciones no varían de forma significativa en las diferentes máquinas. Sin embargo, se debe tener en cuenta que en un ambiente heterogéneo la variedad de compiladores puede llegar a ser igual a la cantidad de máquinas que se utilizan y por lo tanto conocer todos los detalles de los compiladores (procesadores-optimizaciones) es igualmente complejo.

La Figura B.5 y la Figura B.6 muestran los tiempos de ejecución en cada una de las máquinas del CeTAD y del LQT respectivamente. Como en las figuras anteriores donde se muestran tiempos de ejecución, el eje **x** del gráfico corresponde a distintos tamaños de matrices (desde matrices de orden n = 100, hasta matrices de orden n = 1600), y el eje **y** del gráfico muestra el tiempo de ejecución con escala logarítmica.

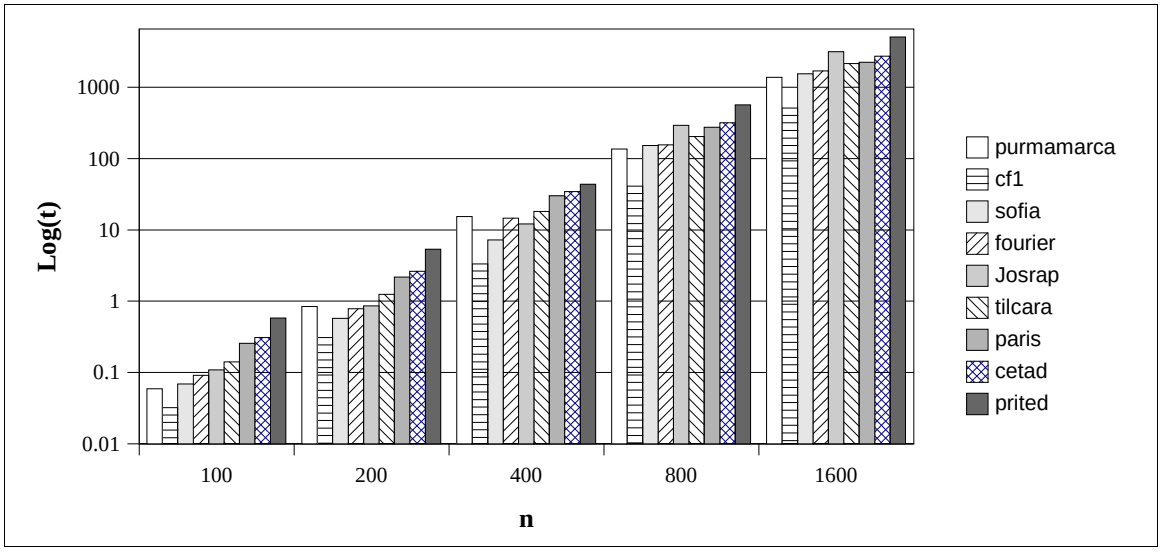

Figura B.5: Tiempos de Ejecución en el CeTAD con Optimización del Compilador.

Comparando la Figura B.5 con la Figura B.1 se puede notar que en muchas computadoras el tiempo total de ejecución disminuye para todos los tamaños de matrices con los cuales se realizaron los experimentos. De la misma manera se puede comparar la Figura B.6 con la Figura B.2. Una vez más, las diferencias relativas de velocidad son difíciles de identificar con precisión por la escala logarítmica en la que se muestran los tiempos de ejecución.

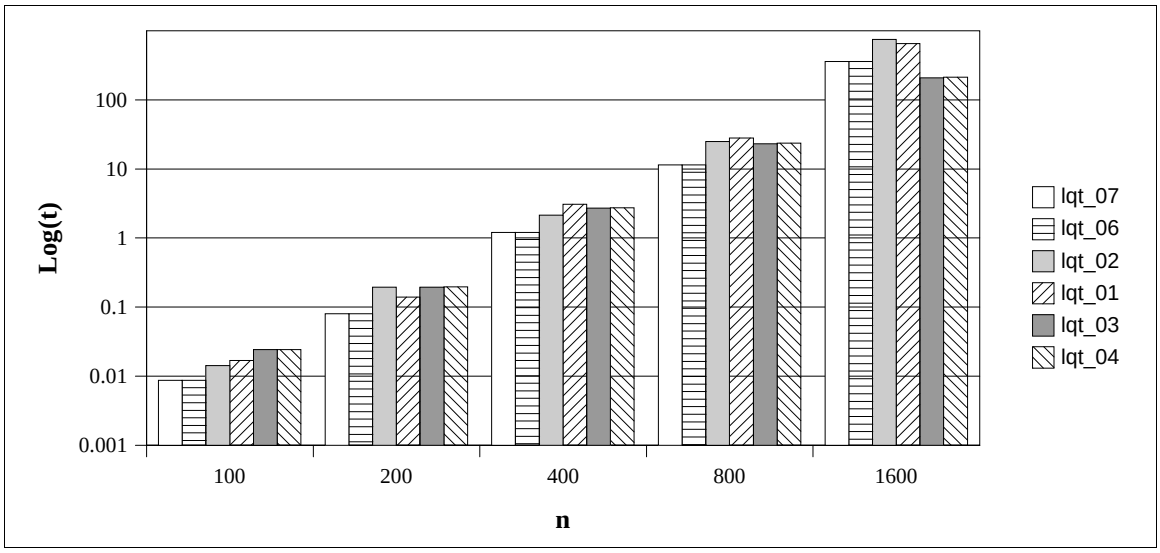

Figura B.6: Tiempos de Ejecución en el LQT con Optimización del Compilador.

La Figura B.7 y la Figura B.8 muestran el rendimiento de cada computadora en Mflop/s, donde se pueden identificar mejor las diferencias con respecto a la ejecución sin ninguna optimización. Dependiendo de la computadora y del tamaño del problema que se resuelve el rendimiento mejora en algunos casos más del 100%: **lqt_01** pasa de poco menos de 50 Mflop/s para *n* = 100 (Figura B.4) a poco menos de 120 Mflop/s para el mismo tamaño de problema (Figura B.8).

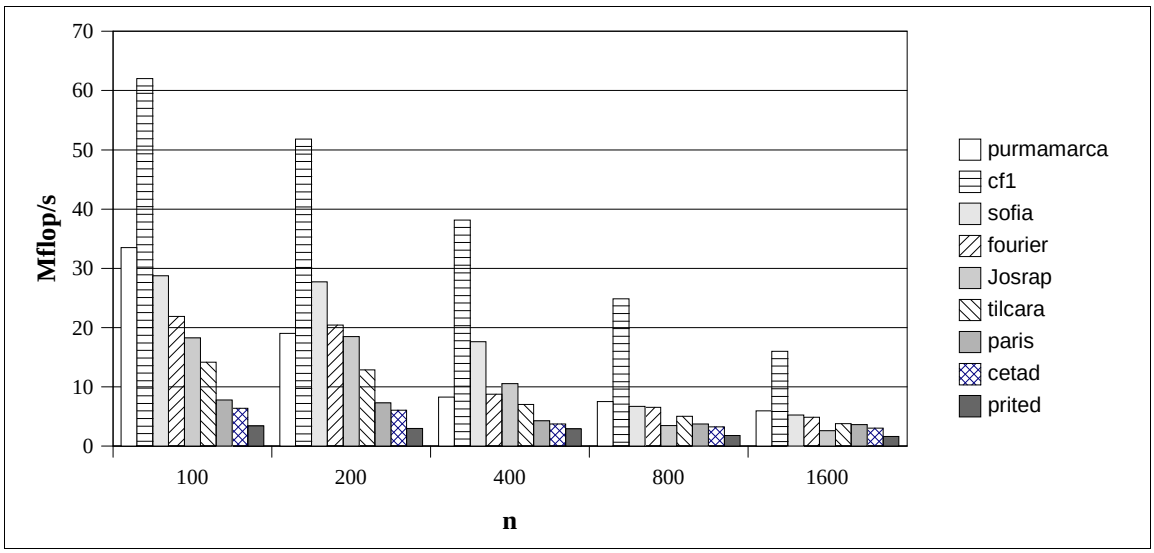

Figura B.7: Mflop/s en el CeTAD con Optimización del Compilador.

Aunque el rendimiento mejora de forma notable en algunos casos como el mencionado y en general mejora en todas las computadoras sigue siendo importante el peso del tamaño del problema en el rendimiento obtenido. De la misma manera, la disminución de rendimiento que tienen las máquinas a medida que aumenta el tamaño del problema tiene distintas características en cada una de ellas y por lo tanto se sigue verificando que las

diferencias de velocidad relativa de las máquinas es dependiente del tamaño del problema que se resuelve.

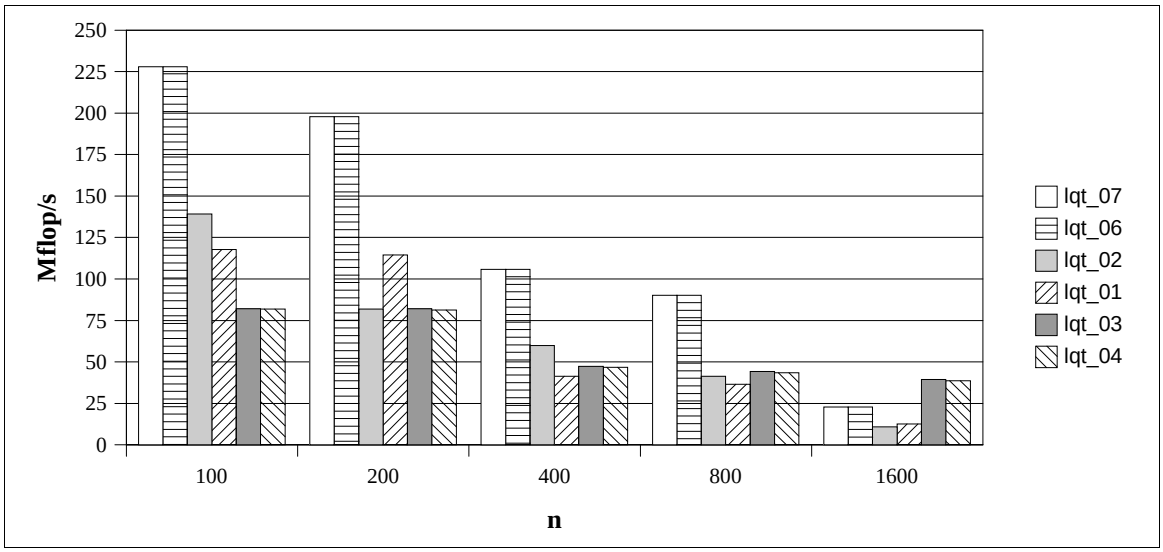

Figura B.8: Mflop/s en el LQT con Optimización del Compilador.

Dado que es muy difícil que un compilador pueda realizar todas las optimizaciones posibles [12], es muy importante contar con código especialmente optimizado para las computadoras que se utilizan. Más ventajoso aún es aprovechar código disponible especialmente optimizado para las computadoras. En la subsección que sigue se muestran los experimentos realizados con este tipo de código y se muestran diferencias muy importantes a nivel de aumento de rendimiento y a nivel conceptual de rendimiento en general de las máquinas que ejecutan código de procesamiento para realizar operaciones provenientes de álgebra lineal.

B.4.3 Rendimiento con Optimización del Código Fuente

La Figura B.9 y la Figura B.10 muestran los tiempos de ejecución de las computadoras del CeTAD y del LQT respectivamente, para cada uno de los tamaños de matrices cuando el código fuente se optimiza para obtener el mejor rendimiento posible. Comparando la Figura B.9 con la Figura B.5 y la Figura B.10 con la Figura B.6 se puede comprobar rápidamente una disminución general del tiempo total de ejecución (en todas las computadoras y para todos los tamaños de problema) muy importante.

A modo de ejemplo, en la Figura B.5 se muestra que la computadora del CeTAD **purmamarca** utiliza aproximadamente un segundo para resolver una multiplicación de matrices de 200×200 elementos. En la Figura B.9 se muestra que el mismo problema en la misma computadora se resuelve en menos de una décima de segundo. Por otro lado, en la Figura B.6 se muestra que las computadoras del LQT **lqt_06** y **lqt_07** utilizan bastante más de cien segundos para resolver una multiplicación de matrices de 1600×1600 elementos. En la Figura B.10 se muestra que el mismo problema en las mismas computadoras se

resuelve en poco más de diez segundos.

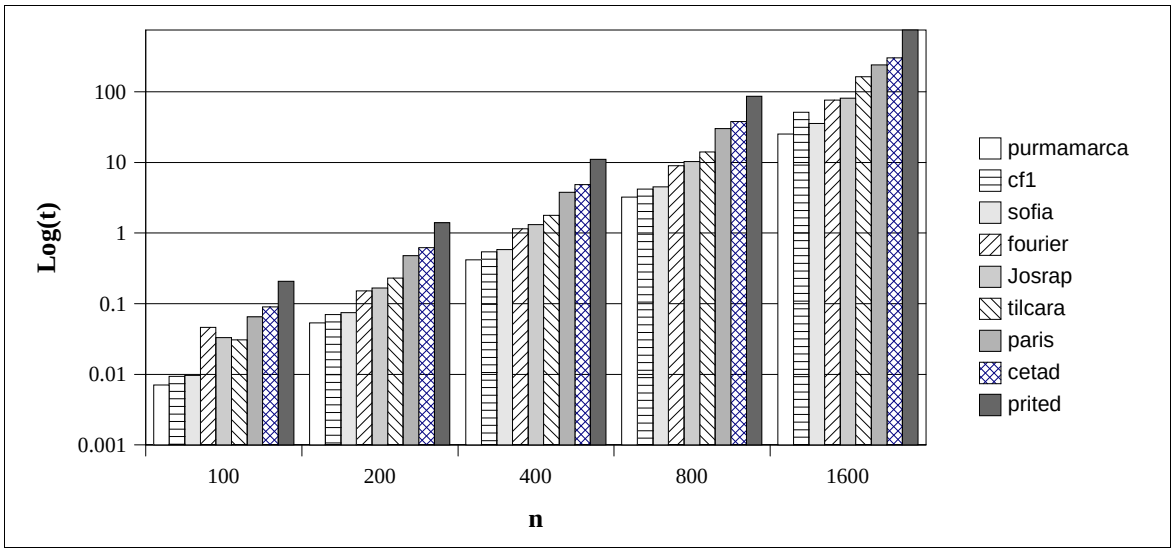

Figura B.9: Tiempos de Ejecución en el CeTAD con Optimización Completa.

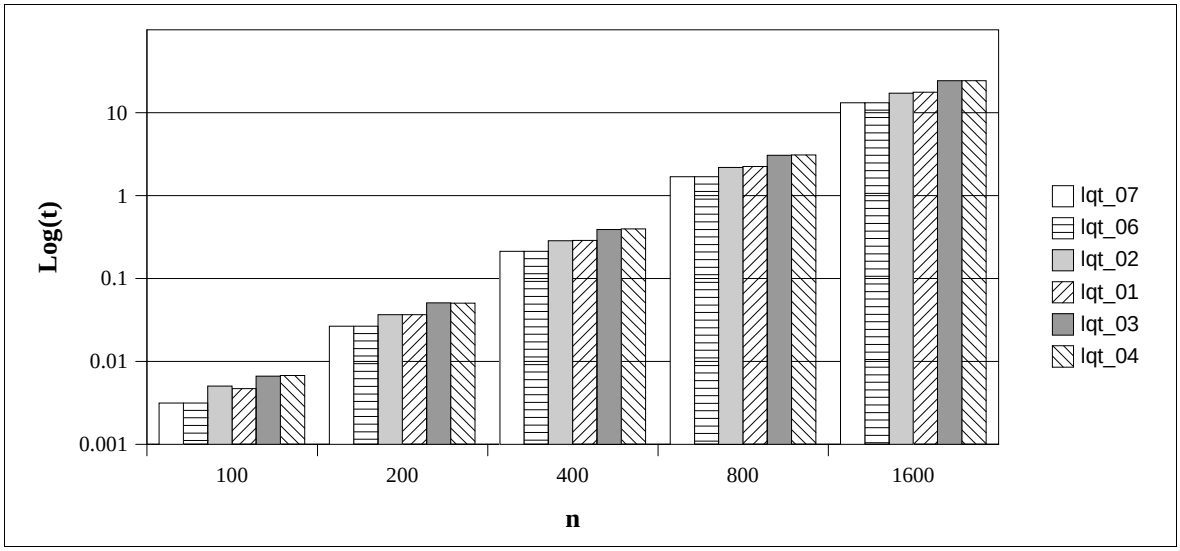

Figura B.10: Tiempos de Ejecución en el LQT con Optimización Completa.

El rendimiento de cada una de las computadoras expresado en Mflop/s se muestra en la Figura B.11 para las máquinas del CeTAD y en la Figura B.12 para las máquinas del LQT. Si se comparan los valores de rendimiento de las máquinas del CeTAD que se muestran en la Figura B.11 (código completamente optimizado) con los de la Figura B.7 (código optimizado por el compilador) y con los de la Figura B.3 (código sin ninguna clase de optimización), las diferencias del rendimiento son muy notables. Estas diferencias no solamente se pueden verificar en cuanto a los valores absolutos, que es idéntico a lo que sucede con el tiempo de ejecución, sino en cuanto a las variaciones de rendimiento dependiendo del tamaño del problema que se resuelve y la relación entre las capacidades relativas de cómputo de cada máquina.

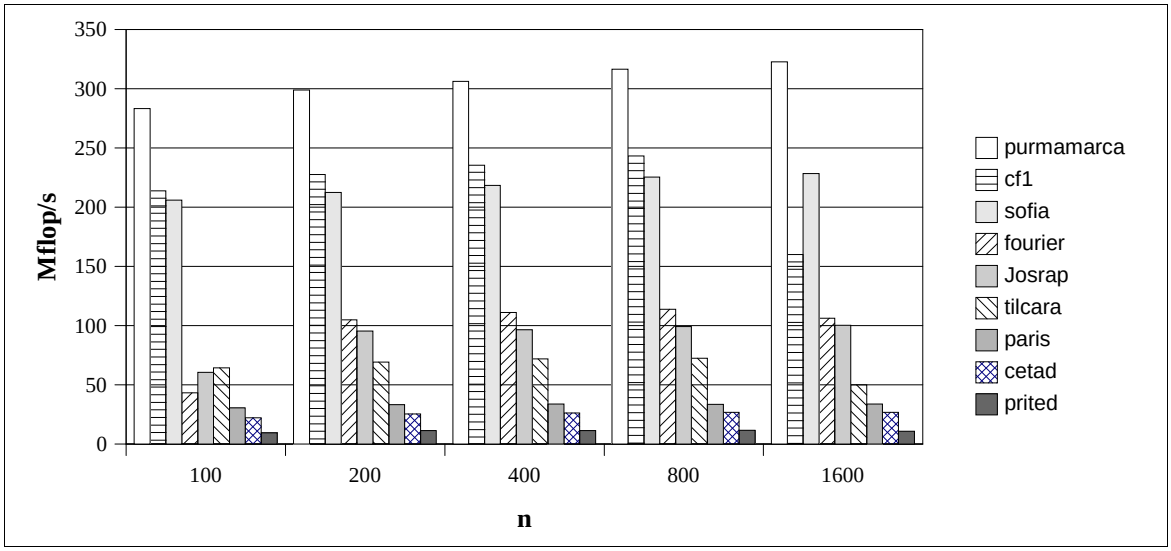

Figura B.11: Mflop/s en el CeTAD con Optimización Completa.

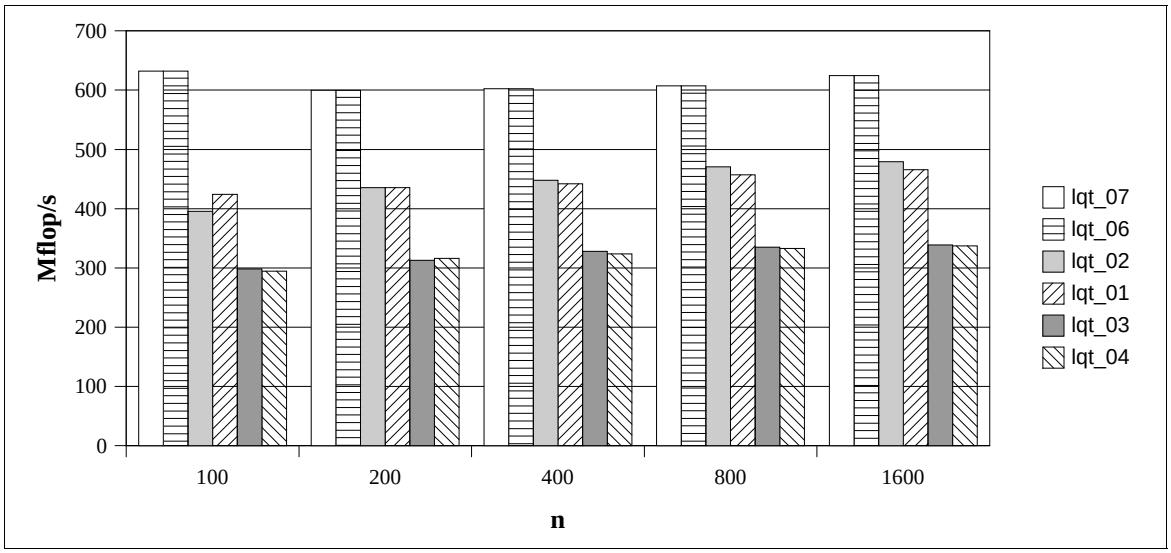

Figura B.12: Mflop/s en el LQT con Optimización Completa.

A partir de estas dos últimas figuras, se pueden identificar características que son casi tan importantes como el aumento de los valores absolutos del rendimiento de todas las máquinas:

El Rendimiento no disminuye a medida que el tamaño del problema crece: las dos figuras lo muestran claramente para casi todas las computadoras. A pesar de la gran heterogeneidad, el rendimiento sigue esta "regla" sin importar procesadores ni ciclos de reloj. Las excepciones que se pueden identificar aparecen en la Figura B.11 y son las computadoras **tilcara** y **cf1**. En ambos casos el rendimiento se degrada notablemente para el valor de n = 1600. En el caso de **tilcara**, pasa de poco más de 70 Mflop/s para n = 800 a 50 Mflop/s (disminución de casi 30% del rendimiento) para n = 1600 y **cf1** pasa de poco

más de 240 Mflop/s a aproximadamente 160 Mflop/s (disminución de casi el 35%) para los tamaños de problema $n = 800$ y $n = 1600$ respectivamente. En ambos casos se debe a la relación entre el tamaño de memoria principal y el tamaño del problema. Para $n = 800$, la cantidad de memoria requerida para las matrices es de poco más de 7 MB y por lo tanto hay espacio suficiente en todas las máquinas, mientras que para $n = 1600$ la cantidad de memoria requerida es de más de 29 MB. Tanto en **tilcara** como en **cf1** (**cetadfomec1**) sucede algo similar: el reporte de actividad del sistema operativo muestra que deben recurrir al espacio de memoria swap aunque su memoria principal es de 32 MB y en principio pueden contener todos los datos que se procesan. Se debe recordar que en la memoria principal se debe contener el kernel del sistema operativo (Linux), partes de memoria que no es posible desalojar al espacio de swap (buffers del sistema operativo, por ejemplo), y un mínimo de código de la aplicación misma. Además, dado que la potencia de cálculo del procesador de **cf1** (Celeron 300 MHz) es más de tres veces mayor que la potencia de **tilcara** (Pentium 133 MHz), es de esperar que el impacto en la disminución del rendimiento sea mayor.

El rendimiento es casi constante o aumenta levemente cuando el problema es mayor. Se puede comprobar en ambas figuras con las excepciones mencionadas, para cada una de las computadoras (una vez más, independientemente de los procesadores y de la heterogeneidad de las máquinas). La base para el cumplimiento de esta regla es el procesamiento por bloques que se realiza sobre los datos. Cada vez que se asigna un dato en memoria cache es reutilizado al máximo, dado que el patrón de acceso a los datos (en memoria) se codifica especialmente con este fin. Esto implica que aumentar la cantidad de datos no aumenta la cantidad de fallos en memoria cache, y más aún, aumentar la cantidad de datos en el caso de las operaciones matriciales y en particular en las multiplicaciones de matrices implica que se puede aumentar la cantidad de accesos a memoria cache con los mismos datos. Es esta manera se puede explicar el incremento en el rendimiento cuando el tamaño de las matrices crece, ya que la asignación de datos a cache sigue siendo un acceso a memoria principal pero una vez en memoria cache se puede reutilizar más veces (aumentando de esta manera la cantidad de “hits” de memoria cache) porque hay más operaciones a realizar por cada dato.

La relación entre las velocidades de cálculo de las computadoras es casi constante, sin variaciones mayores al 10%. Esta característica se puede identificar como una consecuencia del rendimiento casi constante o aumento leve del rendimiento que se explicó anteriormente, pero en el contexto de las aplicaciones paralelas que se resuelven en computadoras heterogéneas es muy importante y necesariamente se debe analizar. Por ejemplo, en la Figura B.12 se puede notar que la computadora **lqt_02** tiene una capacidad de cálculo aproximada de un tercio mayor que lqt_03 en el procesamiento de las matrices de 100×100, y no sólo esto sino que esta relación de velocidades se mantiene aproximadamente constante de manera independiente de los tamaños de matrices que se procesan.

Desde el punto de vista de las aplicaciones paralelas esta última característica que se nota en la experimentación implica una simplificación notable en la forma en que se resuelve el balance de carga computacional. Si se acepta un desbalance de carga computacional del 10% por ejemplo, la forma en que se distribuye la carga se puede implementar de manera independiente del tamaño del problema que se debe resolver en cada computadora, y por lo

tanto se elimina un parámetro de sintonización o al menos se disminuye notablemente el rango de variación.

B.4.4 Matrices Mayores en las Computadoras Con Mayor Capacidad de Cómputo

Muchas de las aplicaciones de cálculo intensivo normalmente tienden a ocupar una gran cantidad de memoria (procesamiento de grandes volúmenes de datos). Por otro lado, también es de esperar que las aplicaciones aprovechen todos (o la mayoría de) los recursos de las computadoras, en particular la memoria principal disponible. Es por esto que se realizaron experimentos para caracterizar el rendimiento de las máquinas cuando el problema a resolver ocupa toda la memoria e incluso cuando los requerimientos son mayores que la memoria principal instalada y se debe recurrir a la utilización del espacio de *swap* configurado.

Las máquinas elegidas para la realización de estos experimentos fueron las de mayor potencia de cálculo de cada una de las redes, es decir **purmamarca** (Pentium II 400 MHz, 64 MB) del CeTAD y **lqt_07** (Pentium III 1 GHz, 512 MB) del LQT. Los experimentos se diseñaron con dos propósitos, utilizando código completamente optimizado:

- Caracterizar el rendimiento para el mayor tamaño de problema que pueda contener la memoria principal.
- Caracterizar el rendimiento para el mayor tamaño de problema que se pueda resolver (incluyendo espacio de memoria *swap*).

La Figura B.13 muestra el rendimiento de **purmamarca** para distintos tamaños de matrices, incluyendo los que implican la utilización de memoria swap durante los cálculos de la multiplicación de matices $C=A\times B$. A partir de las matrices de orden n = 1600 inclusive se muestra la proporción de memoria principal aproximada que se requiere para contener todos los datos del problema. Por ejemplo para n = 1900, el 65% (aproximadamente) de la memoria principal se utiliza para contener los datos de las matrices.

En la Figura B.13 también se muestra el mayor valor de n para el cual todos los datos pueden residir en memoria principal (n = 2000) sin recurrir a memoria swap durante los cálculos. Es decir que a partir de n = 2200 la computadora debe recurrir al *swapping* de páginas de memoria para resolver el problema. De esta manera se explica la disminución del rendimiento para los valores de n = 2200, 2400, 3000 y 3200 con respecto al rendimiento obtenido con n = 2000.

La Figura B.14 muestra el rendimiento de **lqt_07** para distintos tamaños de las matrices que se multiplican. También se muestra el mayor tamaño que se puede contener completamente en memoria principal (n = 5000) y a partir de n = 4000 inclusive la proporción aproximada de memoria principal necesaria para contener todos los datos del problema.

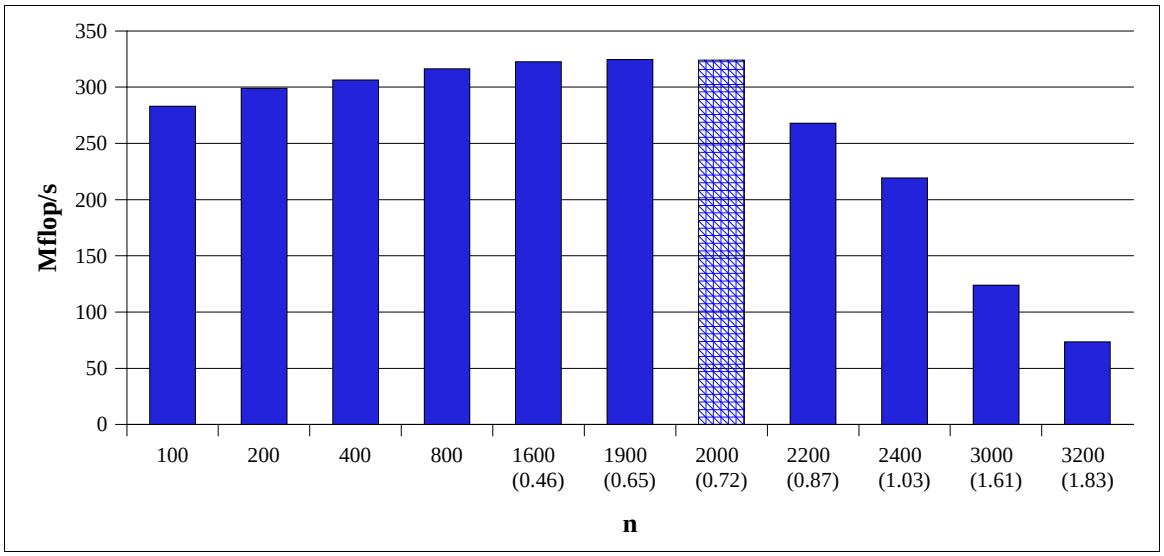

Figura B.13: Mflop/s en **purmamarca**.

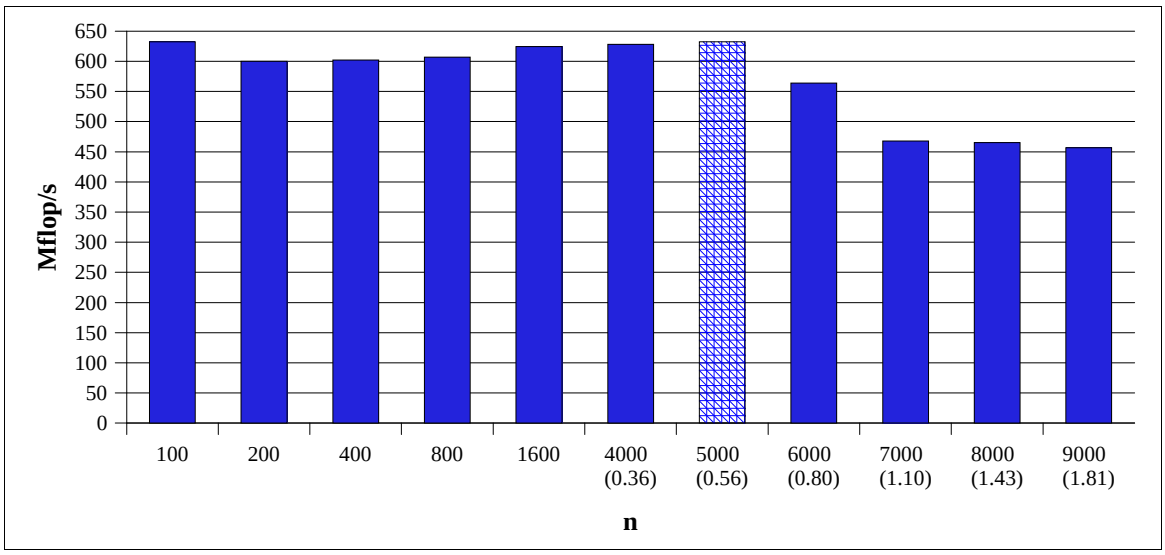

Figura B.14: Mflop/s en **lqt_07**.

En las dos computadoras el rendimiento sigue la misma tendencia: aumenta con el crecimiento de la cantidad de datos del problema mientras la memoria principal es capaz de contener todos los datos a procesar y disminuye a medida que se utiliza mayor proporción de espacio de *swap* (en disco). Sin embargo, las variaciones en **lqt_07** son menores, tanto en lo que se refiere al aumento como a la disminución del rendimiento. Quizás lo más significativo es que el rendimiento no disminuye tanto como en **purmamarca** a partir de la utilización de memoria swap durante los cálculos. Si bien puede ser bastante influyente la velocidad de los discos involucrados, lo más probable es que dos características se combinan para mantener el rendimiento relativamente alto en **lqt_07** aunque se utiliza espacio de memoria swap. Tanto el

- tamaño de las matrices como el
- procesamiento por bloques

se combinan de forma tal que, por un lado la cantidad de operaciones de punto flotante es mucho mayor porque las matrices son mayores y porque los requerimientos de procesamiento son $O(n^3)$, y además el procesamiento por bloques hace que cada dato en memoria principal se utiliza al máximo y por lo tanto se reduce la frecuencia fallos de página que implican el *swapping* de páginas de memoria a disco.

B.5 Rendimiento en las Computadoras del LIDI

La Figura B.15 muestra el rendimiento de las computadoras (homogéneas) del LIDI dependiendo del nivel de optimización de código para la multiplicación de matrices de orden n = 100, 200, 400, 800 y 1600. El rendimiento del código sin ningún tipo de optimización se muestra como "Sin Opt.", el rendimiento del código optimizado por el compilador se muestra como "Compilador" y el rendimiento del programa optimizado a nivel del código fuente para la multiplicación de matrices se muestra como "Completa".

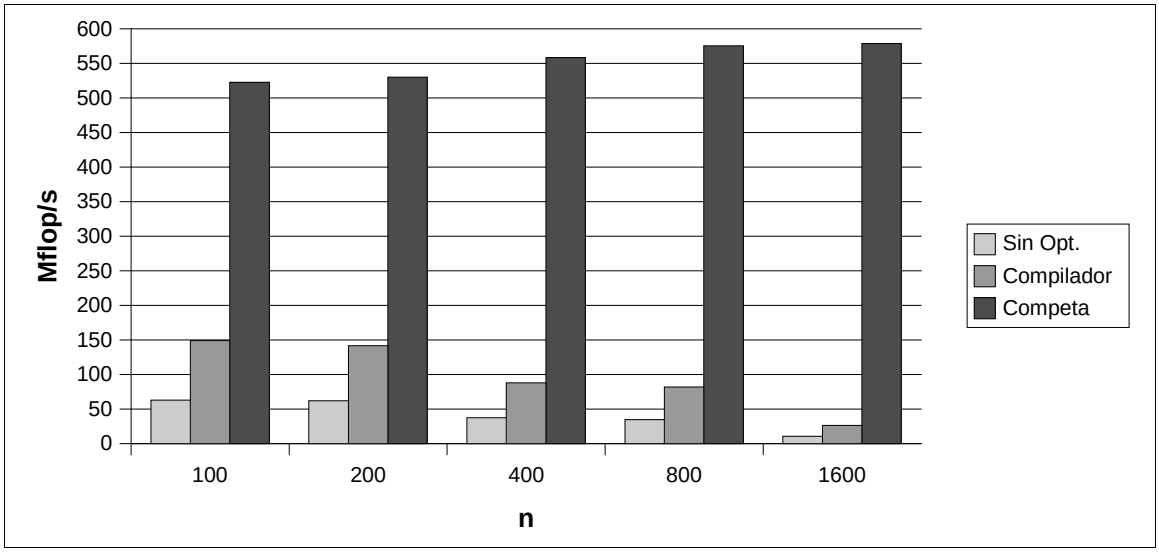

Figura B.15: Rendimiento en Mflop/s Según la Optimización en el LIDI.

De hecho, la Figura B.15 resume lo que sucede en las demás computadoras del CeTAD y del LQT aunque con el valor absoluto de rendimiento propio de las computadoras del LIDI. El peor rendimiento es el que corresponde al código sin ninguna clase de optimización y en el mejor de los casos es de poco más de 50 Mflop/s. Desde el punto de vista del rendimiento el código optimizado por el compilador es aproximadamente tres veces mejor que el no optimizado. Expresado de otra manera, con el código optimizado por el compilador la misma computadora resuelve el mismo problema en un tercio del tiempo que necesita con el código sin optimizar. El rendimiento del código sin optimizar como el del optimizado por el compilador es dependiente del tamaño del problema y se reduce considerablemente a medida que la cantidad de datos a procesar aumenta. El código completamente optimizado es mucho mejor en cuanto a rendimiento que el optimizado por el compilador y es bastante independiente del tamaño del problema, con variaciones que no

superan el 10%. Además, a medida que el tamaño del problema aumenta el rendimiento también aumenta, lo cual es una ventaja considerable con respecto al código sin optimización y al código optimizado solamente por el compilador.

En la Figura B.16 se muestra el rendimiento con código completamente optimizado de las computadoras del LIDI para distintos tamaños de las matrices que se multiplican. Como para las computadoras con mayor capacidad de cálculo del CeTAD y del LQT se muestra el mayor tamaño que se puede contener completamente en memoria principal (n = 2000) y a partir de n = 1600 inclusive la proporción aproximada de memoria principal necesaria para contener todos los datos del problema.

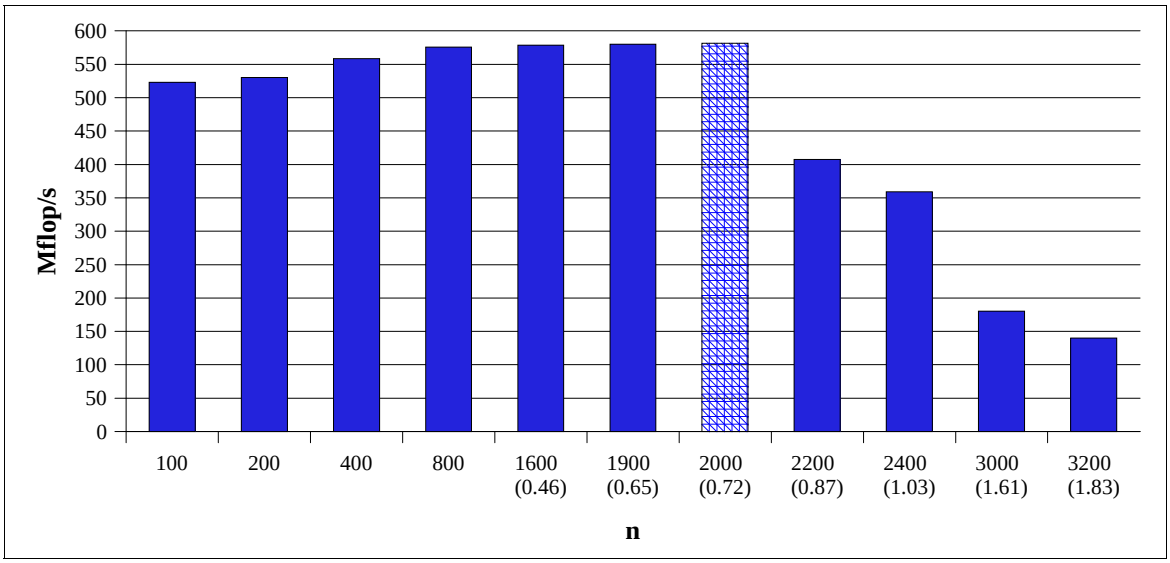

Figura B.16: Mflop/s en las Computadoras del LIDI.

Más allá de las diferencias en valores absolutos, la variación del rendimiento de las computadoras del LIDI es similar a la variación de la máquina con mayor capacidad de cálculo del CeTAD, **purmamarca**, que se muestra en la Figura B.13. De hecho, la gran disminución del rendimiento a medida que se utiliza con mayor frecuencia la memoria swap es similar.

B.6 Conclusiones

A partir de la experimentación realizada, no solamente se tienen valores de rendimiento precisos relacionados con:

- La resolución de la operación elemental de multiplicación de matrices en cada computadora, necesarios para implementar el balance de carga computacional para cómputo paralelo.
- Los valores de rendimiento a utilizar métrica individual de rendimiento de cada una de las mejores computadoras de cada red local, y también para ser utilizado en el cálculo del *speedup* obtenido con la resolución en paralelo.

Además, se llega a que el código a utilizar en cada una de las computadoras debería ser completamente optimizado al menos por dos razones:

1. El rendimiento es varias veces mejor, lo que mejora el tiempo total de ejecución que se requiere de forma significativa (proporcional). El rendimiento secuencial no solamente es crucial para la utilización efectiva de las computadoras (a su máxima capacidad) sino que es sumamente importante para el cálculo correcto de los valores de speedup que se obtienen con procesamiento paralelo.
2. La relación de velocidades de las computadoras es independiente (o con pequeñas variaciones) del tamaño del problema a resolver, lo cual simplifica significativamente la forma en que se resuelve el balance de carga computacional.

Referencias

[1] Alpern B., L. Carter, J. Ferrante, “Space-limited procedures: A methodology for portable high-performance”, International Working Conference on Massively Parallel Programming Models, 1995.

[2] Anderson E., Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users' Guide (Second Edition), SIAM Philadelphia, 1995.

[3] Bilmes J., K. Asanovi□, C. Chin, J. Demmel, “Optimizing matrix multiply using phipac: a portable, high-performance, ansi c coding methodology”, Proceedings of the International Conference on Supercomputing, Vienna, Austria, July 1997, ACM SIGARC.

[4] Dongarra J., J. Du Croz, I. Duff, S. Hammarling, A set of Level 3 Basic Linear Algebra Subprograms, ACM Trans. Math. Soft., 14 (1), March 1988.

[5] Golub G. H., C. F. Van Loan, Matrix Computation, Second Edition, The John Hopkins University Press, Baltimore, Maryland, 1989.

[6] Henning J. L., SPEC CPU2000: Measuring CPU Performance in the New Millenium, Computer, IEEE Computer Society, July 2000.

[7] Hennessy J. L., D. A. Patterson, Computer Architecture. A Quantitative Approach, Morgan Kaufmann, San Francisco, 1996.

[8] Institute of Electrical and Electronics Engineers, IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1984, 1984.

[9] Intel Corporation, Streaming SIMD Extensions - Matrix Multiplication, AP-930, Order Number: 245045-001, June 1999.

[10] McCalpin J. D., M. Smotherman, “Automatic benchmark generation for cache

optimization of matrix algorithms", Proceedings of the 33rd Annual Southeast Conference, R. Geist and S. Junkins editors, Association for Computing Machinery, New York, March 1995.

[11] Saavedra R., W. Mao, D. Park, J. Chame, S. Moon, "The combined effectiveness of unimodular transformations, tiling, and software prefetching", Proceeding of the 10th International Parallel Symposium, IEEE Computer Society, April, 1996.

[12] Whaley R., J. Dongarra, "Automatically Tuned Linear Algebra Software", Proceedings of the SC98 Conference, Orlando, FL, IEEE Publications, November, 1998.

[13] http://developer.intel.com/design/pentiumiii/sml/245045.htm

[14] http://www.netlib.org/atlas

[15] http://www.spec.org