Trabajo Práctico Final

Transformada Rápida de Fourier con Paralelismo

Sistemas Operativos II

Facultad de Ciencias Exactas, Físicas y Naturales
UNC

Mauricio G. Jost

2009

Índice

1. Introducción

Los algoritmos de FFT son múltiples, y existen tantos como cantidad de arquitecturas de computadoras. El presente trabajo pretende mostrar una implementación sencilla, pero no haciendo hincapié en recursos específicos de determinada arquitectura, sino en características más bien generales, presentes en casi todos los sistemas multiprocesadores de la actualidad.

El fin es naturalmente aumentar el grado de utilización de los recursos en esta clase de sistemas, pero sin perder portabilidad. MPI es una herramienta que cumple con estos enunciados, y a la cual se ha recurrido para la implementación final, siguiendo así los lineamientos de la materia.

2. Marco Teórico: FFT

2.1. Usos de DFT y FFT

La DFT posee una gran utilidad, que radica en la posibilidad de aplicar filtrados y otros tipos de procesamiento a señales obtenidas de cualquier tipo de fuente analógica, pero en un dominio de frecuencias. Esto en muchos casos facilita la interpretación del problema por parte del programador, lo cual ayuda a lograr algoritmos más abstractos y de mejores resultados.

Sin embargo, la Transformada Rápida de Fourier (FFT) constituye uno de los cálculos más habituales en el tratamiento de señales digitales. Es una forma eficiente de obtener la Transformada Discreta de Fourier (DFT). Genera el mismo resultado que se obtiene con la DFT. La FFT puede definirse como un reajuste matemático de la DFT para una obtención del mismo resultado a través de menos operaciones matemáticas, y por ello menos tiempo computacional.

2.2. Fundamentos de FFT

Para el procesamiento de N muestras $x_{(n)}$ (N debe ser potencia de 2), se tiene la siguiente ecuación para la obtención de su DFT:

$$X_{(k)} = \sum_{n=0}^{N-1} x_{(n)} e^{-\frac{j2\pi}{N}} \quad , \quad 0 \leq k \leq N-1 \tag{1}$$

El factor

$$e^{-\frac{j2\pi}{N}} = \omega_N^{kn} \tag{2}$$

es llamado *phase factor* o también *twiddle factor*.

Este factor presenta periodicidad, y cierta simetría que no es aprovechada por DFT, pero sí por FFT.

- Propiedad de simetría: $\omega_N^{k+N/2} = -\omega_N^k$
- Propiedad de periodicidad: $\omega_N^{k+N} = \omega_N^k$

Este hecho permite que la cantidad total de multiplicaciones y sumas necesarias para el resultado final, disminuya.

Como consecuencia de la utilización de estas propiedades, se obtiene que la Ecuación 1 pueda expresarse (definida por partes) de la siguiente manera:

$$X_{(2k)} = \sum_{n=0}^{\frac{N}{2}-1} [x_{(n)} + x_{(n+\frac{N}{2})}], \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{3}$$

$$X_{(2k+1)} = \sum_{n=0}^{\frac{N}{2}-1} [x_{(n)} - x_{(n+\frac{N}{2})}], \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{4}$$

La aplicación de las Ecuaciones 3 y 4 a un conjunto de muestras $x_{(n)}$ está representada en la Figura 1. El proceso entero requiere $\log_2 N$ etapas de diezmado (DFT).

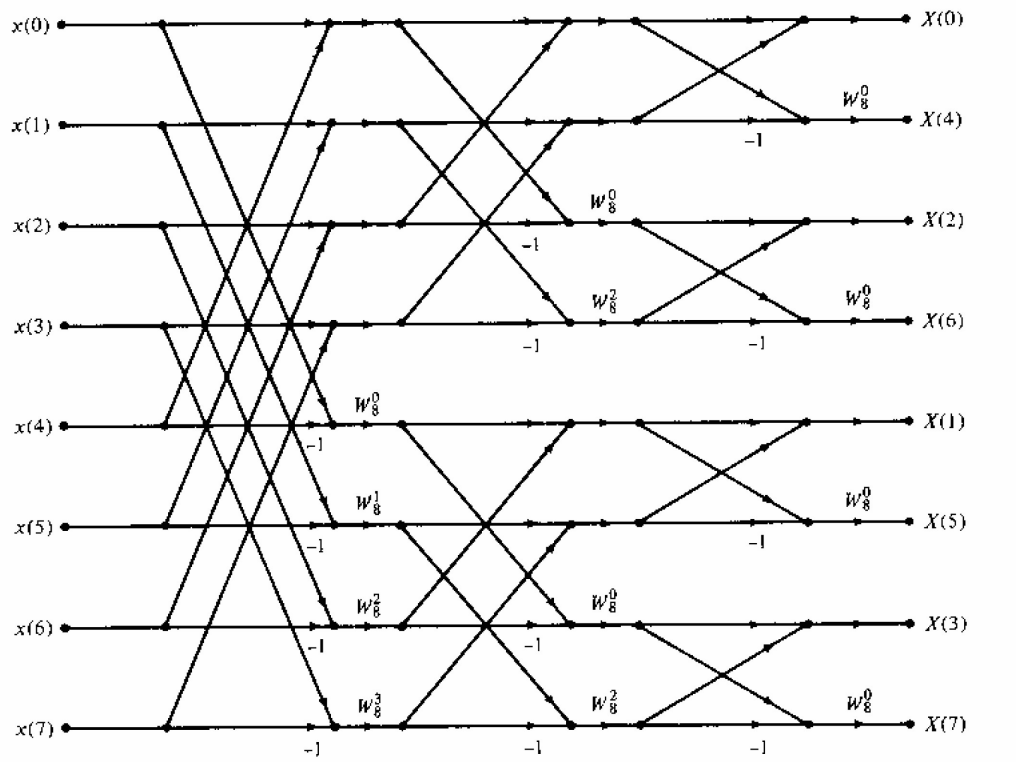

Figura 1: Etapas de la FFT ($N = 8$).

Cada una de las etapas involucra $\frac{N}{2}$ mariposas del tipo mostrado en la Figura (2).

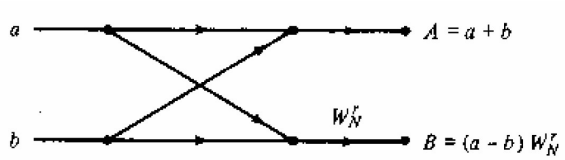

Figura 2: Mariposa para diezmado en frecuencia. Constituye el bloque básico para el algoritmo de FFT.

3. Implementación

3.1. Enfoque del diseño del algoritmo

Si bien el volumen de procesamiento involucrado para obtener la DFT se reduce con el algoritmo de FFT, sigue siendo este altamente costoso en términos de tiempo. En la actualidad existen varias alternativas para implementar la FFT, desde código

en conocidos lenguajes como C, C++ o Fortran, pasando por implementaciones en ensamblador (las más habituales, orientadas específicamente a procesadores DSP), y hasta implementaciones en hardware, con lenguajes como Verilog o VHDL.

No es intención del autor lograr la máxima eficiencia para un sistema dado, sino lograr un algoritmo que tenga buena respuesta general, en cualquier sistema multiprocesador moderno.

3.2. Recursividad

El algoritmo puede ser llevado a una forma recursiva. Así, en cada etapa, se realizan dos llamadas a la misma función de origen. Esta forma de escribir el algoritmo facilita mucho su abordaje conceptual.

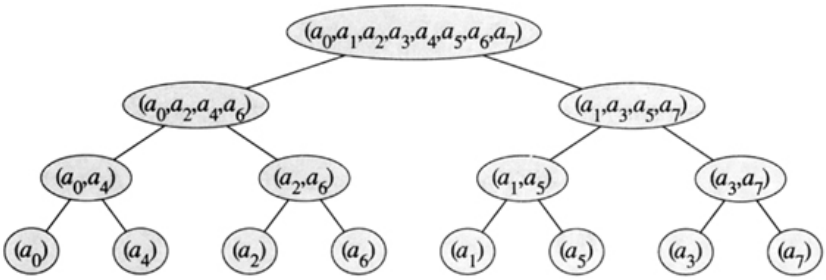

Figura 3: Recursividad de FFT con diezmado en frecuencia. Ejemplo con vector de 8 muestras.

Existen muchas implementaciones escritas en base a bucles, en búsqueda de eficiencia. Por desgracia estas son un tanto más difíciles de desglosar, y su paralelización requiere de un cuidado extremo. En el corriente trabajo se ha optado por algoritmos recursivos, que permiten divisar fácilmente la esencia de la idea sobre la cual fueron construidos.

En la Figura 3 se puede observar el conjunto de llamadas a una función recursiva DFT, y su forma de árbol. Notar que en cada etapa se produce una bifurcación de los datos a nivel de dependencias entre ellos. Esto se ha utilizado para generar paralelismo de grano grueso de sincronización. También es importante destacar que una vez logrado el último nivel de recursividad, los datos requieren de un ordenamiento especial antes de ser reagrupados.

3.3. Paralelismo

Como se puede observar en la Figura 1, no existe dependencia entre dos bloques de datos luego de una bifurcación. Esto permite que los hilos o procesos que llevan a cabo las tareas para cada bloque, no requieran sincronizaciones tan frecuentes, o dicho de otra forma, presenten una granularidad alta de sincronización. Las condiciones de paralelismo de este algoritmo son favorables desde este punto de vista.

Sin embargo, antes de la bifurcación, no es sencillo lograr procesamiento paralelo que brinde mejoras considerables. Esto es a causa de que en etapas iniciales, la granularidad es fina. La interdependencia de los datos es alta, lo cual hace necesario un algoritmo bastante más complejo que las coordine, y de una sobrecarga no menor.

3.3.1. Etapas de DFT con Paralelismo

En este trabajo se ha optado por buscar la eficiencia desde la simpleza y la casi nula sobrecarga de sincronización. Esto implica que el algoritmo no es capaz de distribuir el trabajo entre una cantidad arbitraria de procesadores, sino únicamente para una cantidad N_p donde $N_p = 2^v$ con v entero (en otras palabras, N_p debe ser potencia de 2). De esta forma, se pueden distribuir los trabajos sin una sobrecarga considerable, y hacer que cada tarea tenga un grado de dependencia muy bajo respecto de su tarea par, aquella que surgió de la misma bifurcación. También se logra un balance de carga que mejora con el aumento de la cantidad de muestras (Figura 4).

En la Figura 1 se puede ver que es necesario luego de la última etapa de DFT, realizar un reordenamiento de los datos antes de presentarlos. El método que permite obtener la secuencia de índices es el *Bit Reversal*. El ordenamiento en sí, puede consumir un tiempo importante, y es por ello que se ha optado por paralelizar también su ejecución.

3.3.2. *Bit Reversal* con Paralelismo

El método utilizado para lograr una ejecución más rápida del algoritmo de ordenamiento es sencillo. Consiste en agrupar los datos procesador y desordenados (u ordenados de una forma no útil) luego de todas las etapas de DFT, y redistribuirlos, para que cada procesador reordene una parte determinada del bloque final. Cada procesador tomará datos de todo el conjunto desordenado, según su propio cálculo de *Bit Reversal*, y los agrupará en un bloque que, agrupado de forma contigua con el resto de los bloques procesados por el resto de los procesadores, resultará en el bloque final con el orden deseado.

3.4. Balance de carga

Para obtener cierto balance de carga, se ha optado por una *asignación parcial de tareas*. En el momento en que todas las mariposas de una cierta etapa han sido calculadas, se procede a asignar cada bloque resultante (de dos), a diferentes procesos. La regla a seguir es sencilla:

1. Siempre parte con el conjunto de mariposas de la etapa 1° el proceso *root*.
2. Llegado el momento de la bifurcación, se reasignan tareas. El bloque superior (siguiendo la Figura 1) se mantiene en el proceso actual. El bloque inferior, en

caso de haber aún procesos ociosos, es asignado a uno de ellos. La regla de asignación está asociada a la correspondencia que tiene cierto proceso, a cierto *bloque básico de asignación*[1].

3. Una vez que se han ocupado todos los procesos existentes, las asignaciones permanecen sin modificación.

La ecuación utilizada para obtener el proceso responsable de tratar cierto bloque está dada por:

$$P_{id} = B_{inicio} \frac{N_P}{N_M} \tag{5}$$

donde:
P_{id} es el número de proceso.
B_{inicio} es el punto inicial del bloque.
N_P es la cantidad de procesos.
N_M es la longitud de la muestra (en coeficientes).

La Figura 4 muestra un ejemplo.

[1] Ejemplo: se desea conocer el procesador asignado al bloque con inicio en 192. En caso de tener $N_P = 8$ procesadores y $N_M = 256$ muestras, la regla asigna al bloque con inicio en 192 el procesador $192 \frac{N_P}{N_M} = 192 \frac{8}{256} = 6$.

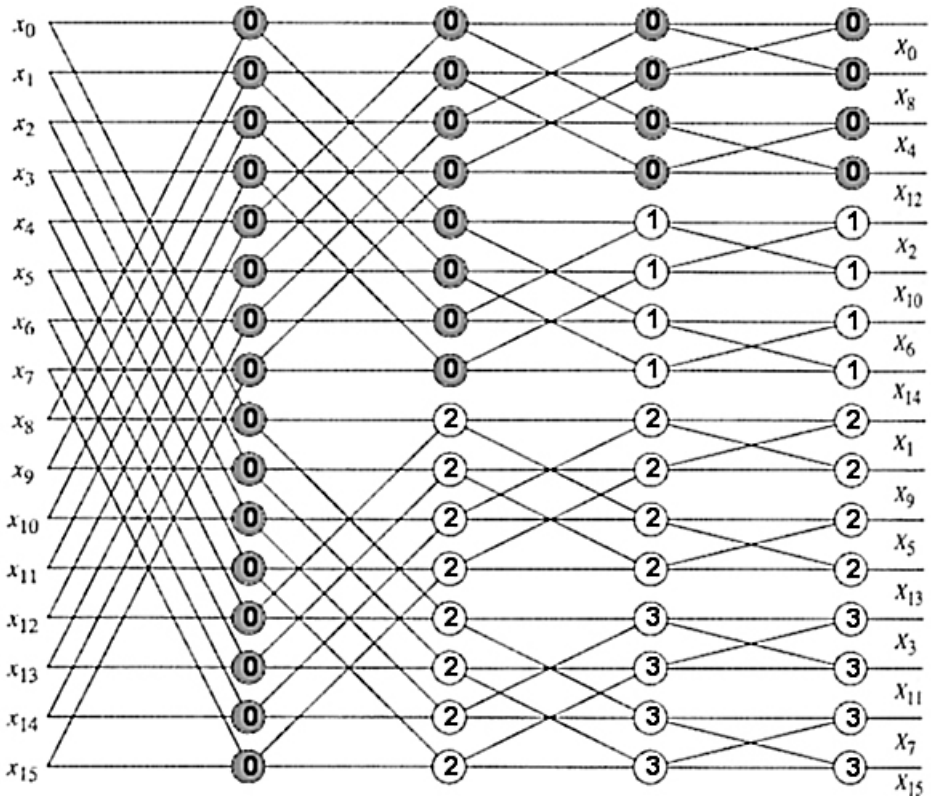

Figura 4: Ejemplo de balance de carga para el caso de $N_p = 4$ y $N_{muestras} = 16$. Con tono más oscuro se pueden observar las asignaciones al proceso *root*. Cada círculo representa un dato calculado por el procesador indicado. En este caso la *unidad mínima de distribución de trabajo* es $\frac{N_{muestras}}{N_p} = 4$.

Nótese en la Figura 4 que el algoritmo es imparcial al inicio. A medida que se avanza en las etapas la carga es mejor balanceada. Al momento de alcanzar la *unidad mínima de distribución*, el trabajo deja de distribuirse, y cada procesador continua con la totalidad de las DFT asociadas al sub-bloque, y a sus sub-bloques.

En la etapa de ordenamiento *Bit Reversal*, la totalidad de los datos son repartidos a cada procesador. Cada uno de ellos utiliza ciertos elementos del vector desordenado, para generar un bloque de lo que será el vector final.

4. Mediciones

Las mediciones tomadas muestran tiempos en [Seg*[2]] para diferentes tamaños de vector, y para diferente cantidad de procesadores. El sistema utilizado para todos los casos es una PC Notebook HP530, CoreDuo 1.66 GHz, 2GiB RAM, SO Fedora 10.

[2] *Seg** es la unidad de referencia utilizada. Resulta del retorno de la función ANSI C *clock_t clock()* dividido entre *CLOCKS_PER_SEC*. Es usada a los efectos de establecer comparaciones entre dos algoritmos, y no se pretende con ella sugerir tiempos absolutos de ejecución.

Muestras	16Ki	32Ki	64Ki	128Ki	256Ki	512Ki	1Mi	2Mi	4Mi	8Mi	16Mi	32Mi	64Mi
1 uP	0.01	0.02	0.05	0.13	0.31	0.72	1.54	4.41	7.30	16.62	34.90	70.57	154.36
2 uP	0.01	0.02	0.04	0.12	0.21	0.45	0.92	2.00	4.33	9.12	19.46	38.68	>10min

Cuadro 1: Mediciones. Tiempos en [seg*].

La aparentemente inexplicable medición obtenida para 2uP y 64Mi muestras, es causada por falta de memoria principal. Para ciertas etapas del algoritmo, cada proceso requiere volúmenes grandes de memoria (por ejemplo al hacer *Bit Reversal*). No disponer de esa memoria implica la utilización de memoria de intercambio (memoria secundaria), y por ende aumentos imprevistos de tiempo de ejecución.

La Figuras 5 y 6 muestran estos resultados.

Las Figuras 7 y 8 expresan la relación entre resultados para 1 y 2 procesadores.

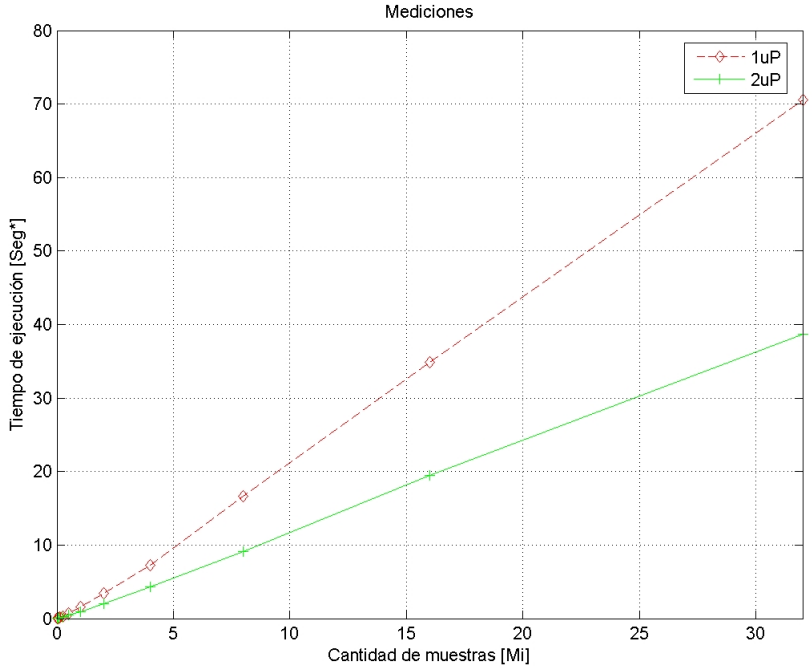

Figura 5: Mediciones de la FFT con paralelismo.

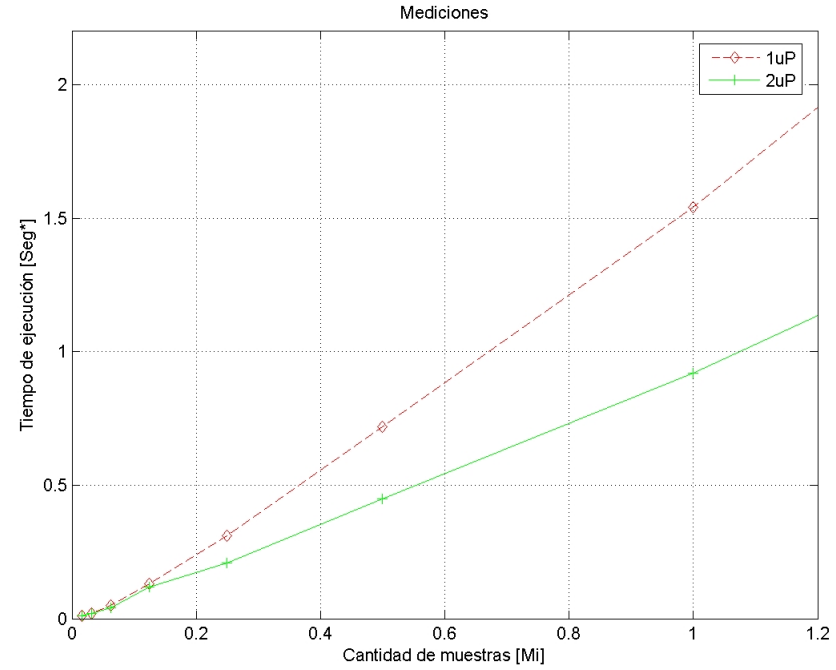

Figura 6: Mediciones de la FFT con paralelismo. Para una cantidad de muestras pequeña, las ventajas son menos evidentes.

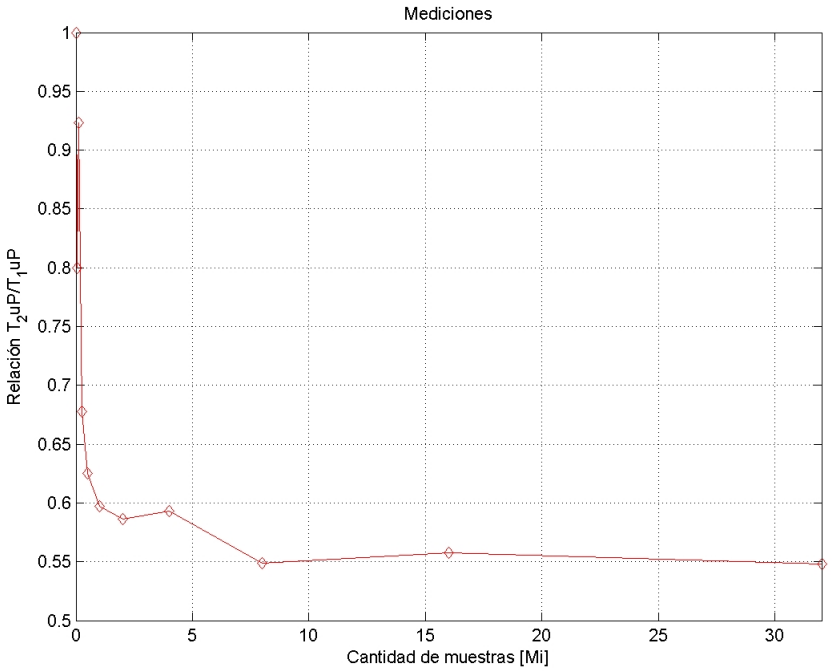

Figura 7: Relación entre mediciones de la FFT con paralelismo sobre 1 y 2 procesadores.

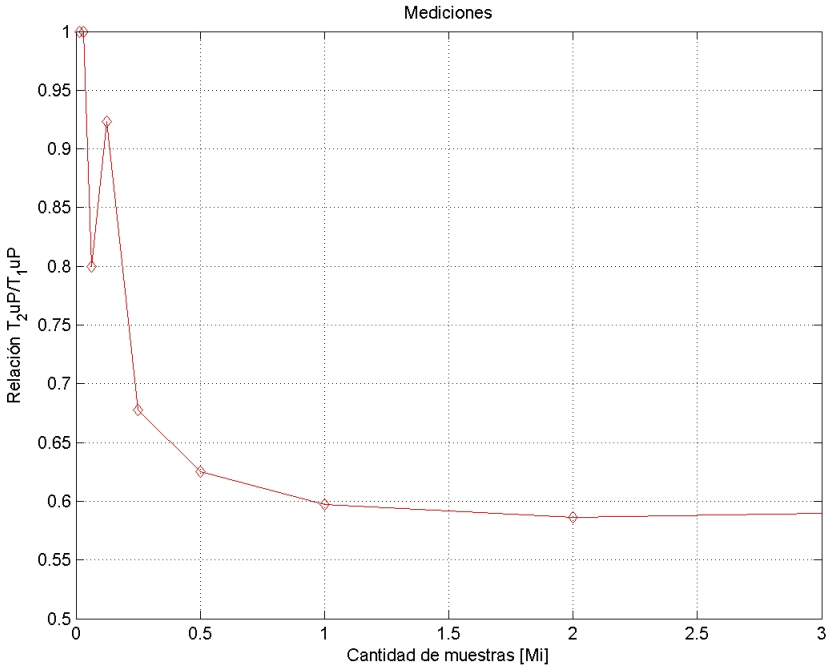

Figura 8: Relación entre mediciones de la FFT con paralelismo sobre 1 y 2 procesadores. Ventajas pobres para cantidad reducida de muestras.

Referencias

[1] Carleton University, Mohammed Aboul-Magd, *Recursive Fast Fourier Transform Algorithm*, http://www.scs.carleton.ca/~maheshwa/courses/5703COMP/Seminar08/mohammed-reprt.pdf. Pg. 5.

[2] National Taiwan University, *Fast Fourier Transform (FFT)*, http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html.

[3] FSU Computer Science, *Efficient FFT Implementations*, http://www.cs.fsu.edu/~cop4531/slideshow/chapter32/32-3.html.

[4] Proakis, J. G. y Manolakis, D. G., 2003. *Tratamiento Digital de Señales*. Tercera Edición. Prentice Hall. Madrid.

[5] Cátedra de DSP - FCEFyN (UNC). Apunte *Transformada de Fourier y el Algoritmo FFT*, http://www.dsp.efn.unc.edu.ar/documentos/FFT.pdf.