Análisis del desempeño de aplicaciones paralelas con openMP en dispositivos móviles multicore, caso de estudio: multiplicación de matrices

MTI Jimmy Josué Peña Koo

Instituto Tecnológico Superior del Sur del Estado de Yucatán: ITSSY

jimjpk@gmail.com

Dr Luis Fernando Curi Quintal

Facultad de Matemáticas de la Universidad Autónoma de Yucatán: UADY

cquintal@uady.mx

Resumen

El presente trabajo analiza el desempeño de procesadores multicore en dispositivos móviles al ejecutar una aplicación paralela implementada con OpenMP y C. La arquitectura multicore ha sido la respuesta de los fabricantes de microprocesadores a los problemas de eficiencia energética que se presentan al incrementar la frecuencia del reloj para incrementar el desempeño de procesadores de un solo núcleo. Esta arquitectura reúne varias unidades de procesamiento energéticamente eficientes en un solo microprocesador. Sin embargo para explotar el potencial del conjunto de núcleos, las aplicaciones deberán diseñarse bajo el paradigma de computación paralela. Se aplicó una metodología de programación multi-hilos, propuesta por Intel, para la implementación de una aplicación que multiplica matrices en paralelo. Esta aplicación se ejecutó en tres diferentes dispositivos móviles. Los resultados obtenidos muestran un incremento en el desempeño de la aplicación al incrementar el número de núcleos

que participan en la ejecución, con un nivel de eficiencia del sistema de al menos el 88% en un procesador quad-core.

Palabras Clave: Android, NDK, OpenMP, Programación Paralela.

Abstract

This paper analyzes the performance of multicore processors in mobile devices in order to execute a parallel application implemented with OpenMP and C. The multicore architecture has been the response of microprocessor manufacturers to energy efficiency issues that arose when the clock frequency is increased to improve the performance of single-core processors. This architecture combines several energy-efficient processing units in a single chip. However, applications must be designed following the parallel computing paradigm to maximize processor performance. A parallel matrix multiplication application was deployed using a methodology of multi-threading programming proposed by Intel. This application was executed in three different mobile devices. The results show an improvement in application performance when the number of cores is increased, and having a level of system efficiency of at least 88% on a quad-core processor.

Keywords: *Android, NDK, OpenMP, Parallel Programming.*

1. Introducción

Las arquitecturas de computadoras han buscado, una mayor velocidad del procesador, más memoria caché (adjunta al procesador), más memoria principal y una mayor velocidad en la comunicación entre la memoria y el procesador, tal como lo predijo la Ley de Moore en 1965, en la cual se indica que el número de transistores en un circuito integrado se duplica aproximadamente cada dos años. Sin embargo, el incremento en el consumo de energía y el calor que es necesario disipar al incrementar la frecuencia

del reloj del procesador y la densidad en los circuitos integrados han llegado a su límite físico para micro-procesadores que constan de una única unidad de procesamiento.

Los procesadores inicialmente fueron desarrollados con un solo núcleo. La empresa Rockwell International fundada en 1973 por Willard Rockwel, la cual tenía sus orígenes en automovilismo, fue la primera en fabricar procesadores de dos núcleos a mediados de la década de los 80's.

La tecnología de semiconductores todavía hace honor a la Ley de Moore; este incremento se usa ahora para poner más núcleos en el mismo procesador central, en lugar de tratar de seguir incrementando la frecuencia del reloj, permitiendo que los accesos a memoria caché dentro del núcleo sean eficientes. Esta arquitectura multicore presenta el reto de implementar las aplicaciones usando procesamiento paralelo, con miras de explotar al máximo posible el potencial del procesador.

Un dispositivo móvil con procesador multicore es una computadora de bolsillo con más de un núcleo, que puede llevar a cabo diferentes funciones. La arquitectura multicore permite que las aplicaciones que requieren tiempo elevado de cómputo puedan ejecutarse en menor tiempo, siempre y cuando se utilice la mayor cantidad de núcleos para realizar dicho procesamiento.

Actualmente existen muchos y muy variados lenguajes de programación, de los cuales no todos tienen la capacidad de aprovechar al máximo los recursos de los equipos con procesadores multicore.

A pesar de que los dispositivos móviles son producidos con procesadores de varios núcleos, las aplicaciones de software aún siguen un modelo de ejecución secuencial similar al de los procesadores de un solo núcleo. Por esta razón, el presente trabajo utiliza las ventajas que el hardware proporciona para mejorar el rendimiento y la eficiencia de las aplicaciones por medio de la programación paralela.

Para el estudio del comportamiento de sistemas paralelos en dispositivos móviles, se desarrolló una aplicación para el Sistema Operativo Android, la cual realiza

multiplicaciones de matrices de dimensión variable proporcionada por el usuario. Para el desarrollo de la aplicación se utilizó el Paquete de Desarrollo Nativo de Android (NDK, Native Development Kit) que es un conjunto de herramientas que permiten implementar parte de las aplicaciones usando código nativo de lenguajes como C o C++, compilada con OpenMP (Open specifications for Multi Processing) una API para programar aplicaciones paralelas multi-hilos de manera explícita.

Para el análisis del desempeño de las aplicaciones móviles secuenciales contra las paralelas se empleó el paralelismo de datos bajo la metodología de programación multi-hilos; adicionalmente, al tiempo de ejecución como medida de rendimiento, también se analizaron las métricas de aceleración (speed-up) y eficiencia.

2. Métodos

La investigación se basó en el análisis del desempeño de una aplicación móvil la cual genera matrices usando memoria dinámica de tamaño variable *n* definido por el usuario desde la interfaz, para su posterior multiplicación (AxB=C). Cada elemento de la matriz A se inicia con $A[i,j]=i+j$ y la matriz B será la matriz identidad. De forma tal que la suma de los elementos de la matriz resultante (C) es $n^2(n-1)$.

El desarrollo de la aplicación se realizó siguiendo el esquema de la Metodología de Programación Multi-hilos, la cual opera con un ciclo de desarrollo genérico propuesto por Intel Software College, consistiendo en cuatro etapas: Análisis (buscar código donde se realiza cómputo intensivo), Diseño (introducir hilos, determinar cómo implementar una solución paralelizada), Depuración (detectar cualquier problema como resultado de usar hilos) y Afinación para mejorar el rendimiento (lograr el mejor rendimiento en paralelo), tal como se puede observar en la figura 1.

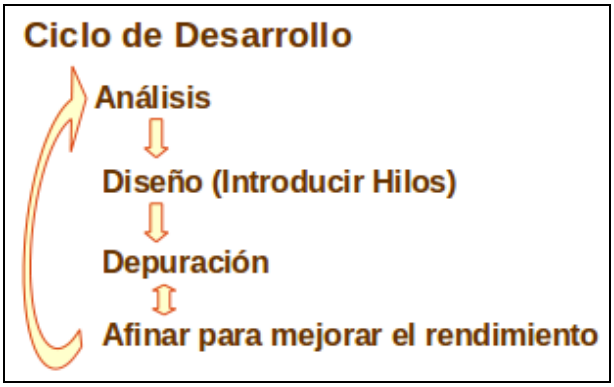

Figura 1. Ciclo de desarrollo de metodología de programación paralela.

Análisis. Con la finalidad de encontrar las regiones que consumen mayor tiempo de procesamiento se compiló el algoritmo secuencial para su análisis empleando la herramienta GNU gprof, por ser una herramienta de código abierto que ayuda a identificar los fragmentos de código que consumen más tiempo de lo esperado, rastrea el flujo del programa, ayudando así a la depuración y a resolver problemas de rendimiento. Se ejecutó el proceso secuencial para matrices de tamaño 300, 600, 900 y 1200, resultando que la función multiplicarMatrices es la que consume el mayor tiempo tal como se puede observar en la tabla 1, por lo que es factible paralelizar la función.

Tabla 1. Resultado del análisis del con GNU gprof.

Dimension	% Time	Cumulative seconds	Self seconds	Name
300	100%	0.18	0.18	multiplicarMatrices
	0.00%	0.18	0.00	iniciarMatrices
600	99.25%	1.32	1.32	multiplicarMatrices
	0.75%	1.33	0.01	iniciarMatrices
900	99.74%	7.67	7.67	multiplicarMatrices
	0.26%	7.69	0.02	iniciarMatrices
1200	99.54%	10.83	10.83	multiplicarMatrices
	0.46%	10.88	0.05	iniciarMatrices

Diseño. Se empleó el paralelismo por datos por medio de la herramienta OpenMP, cuyas construcciones son directivas de compilación o *pragmas* basadas en memoria compartida, con el objetivo de paralelizar la multiplicación de matrices. En su construcción se definió el alcance de las variables, de tipo shared (compartidas por todos los procesos) y de tipo private (cada proceso tiene una copia de la variable). Se diseñó paralelizando ciclos, donde el master crea los hilos adicionales que cubren todas las iteraciones del ciclo (un hilo por núcleo), dado que el algoritmo no existen dependencias entre las iteraciones. Al término de esta etapa se pudo obtener el código tal como se puede observar en la figura 2.

```
hilos = omp_get_max_threads();
#pragma omp parallel num_threads(hilos) shared(a,b,c,total) private(i,j,k,sum)
{
  #pragma omp for
  for (i=0; i < dimension; i++){
    for (j=0; j < dimension; j++){
      sum = 0;
      for (k=0; k<dimension; k++)
        sum += (a[i * dimension + k] * b[k * dimension + j]);
        total  += sum;
      c[i * dimension + j] = sum;
    }
  }
}
```

Figura 2. Código resultante del diseño, paralelizado con OMP.

Depuración. Durante esta etapa se observó que los resultados eran diferentes en cada ejecución, por lo que fue necesario manipular las concurrencias por medio de la declaración de secciones críticas al actualizar la variable compartida total, con la finalidad de asegurar la exclusión mutua en la ejecución del bloque, es decir, que no pueda ser ejecutado de forma simultánea por más de un hilo, tal como se puede observar en la figura 3.

```
hilos = omp_get_max_threads();
#pragma omp parallel num_threads(hilos) shared(a,b,c,total) private(i,j,k,sum)
{
  #pragma omp for
  for (i=0; i < dimension; i++){
    for (j=0; j < dimension; j++){
      sum = 0;
      for (k=0; k<dimension; k++)
        sum += (a[i * dimension + k] * b[k * dimension + j]);
      #pragma omp critical
      {
        total  += sum;
      }
      c[i * dimension + j] = sum;
    }
  }
}
```

Figura 3. Código resultante de la depuración, paralelizado con OMP.

Afinar para mejorar el rendimiento. En esta última etapa el objetivo fue generar una distribución adecuada del trabajo en paralelo por núcleo por medio del paralelismo explícito. Se requería que el algoritmo paralelo especifique explícitamente cómo cooperan los procesadores para la multiplicación de matrices en el dispositivo móvil, tomando en cuenta el número de núcleos que tuviese. Como resultado se obtuvo el código a observar en la figura 4.

```
hilos = omp_get_max_threads();
#pragma omp parallel num_threads(hilos) shared(a,b,c,total,i) private(j,k,m,sum,ini,fin)
{
  #pragma omp for
  for (i=0; i < hilos; i++){
    ini = i*dimension/hilos;
    fin = ((i+1)*dimension/hilos)-1;
    for (m=ini; m<=fin; m++){
      for (j=0; j < dimension; j++){
        sum = 0;
        for (k=0; k<dimension; k++)
          sum += (a[m * dimension + k] * b[k * dimension + j]);
        #pragma omp critical
        {
          total  += sum;
        }
        c[m * dimension + j] = sum;
      }
    }
  }
}
```

Figura 4. Código resultante de afinar para mejorar el rendimiento.

3. Resultados

A partir del algoritmo paralelizado y afinado se construyó la interfaz gráfica para la aplicación android con NDK para poder hacer las llamadas a la API OpenMP, obteniendo la aplicación tal como se puede observar en la figura 5.

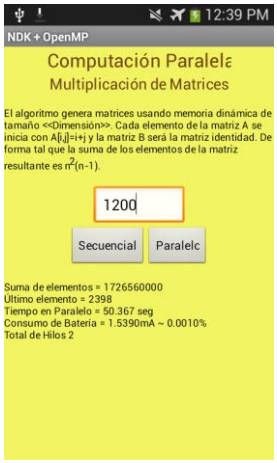

Figura 5. Aplicación android con NDK y OpenMP.

Se analizó el desempeño de la aplicación en tres dispositivos móviles: GT-P5110(Tablet Samsung Galaxy Tab 2 10.1), GT-S7582L (Celular Samsung Galaxy S Duos) y K011(Tablet Asus Memo 8), cuyas características se describen en la tabla 2.

Tabla 2. Dispositivos móviles empleados en el análisis.

Modelo	Procesador	Memoria
GT-P5110	ARM Cortex-A9 1000Mhz Dual-Core	1Gb
GT-S7582L	Cortex A9 Processor 1.2 Ghz Dual-Core	1Gb
K011	Intel Atom Z3745 Quad-core 1.33 Ghz	1Gb

Para cada dispositivo móvil se midió el tiempo de respuesta de ejecución en segundos, para la multiplicación de matrices de tamaño 300, 600, 900 y 1200, obteniendo los valores observados en la tabla 3.

Tabla 3. Resultado en segundos de tiempos de ejecución por dispositivo.

	300		600		900		1200	
	Secuencial	**Paralelo**	**Secuencial**	**Paralelo**	**Secuencial**	**Paralelo**	**Secuencial**	**Paralelo**
GT-P5110	0.553	0.396	6.650	4.299	31.306	20.078	101.958	64.859
GT-S7582L	0.492	0.370	5.638	4.056	25.318	16.917	70.318	50.367
K011	0.265	0.156	3.427	1.309	13.000	4.163	31.944	10.136

De acuerdo al comportamiento gráfico presentado en la figura 6, podemos observar la relación directa que existe entre el tiempo de ejecución con respecto al número de hilos (se ejecutó un hilo por núcleo). A mayor número de hilos se minimizó el tiempo de ejecución. Sin olvidar la ley de Amdahl que indica que la eficiencia obtenida en una implementación paralela viene limitada por la fracción del programa no paralelizable.

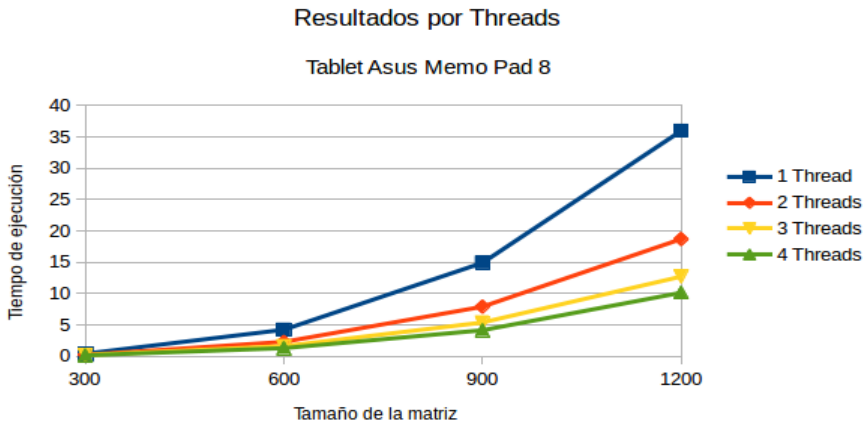

Figura 6. Tiempos de ejecución por hilos.

El análisis del rendimiento basado en la aceleración (Speed-Up) de una aplicación paralela es determinar cuántas veces más rápida es una aplicación paralela con respecto a una secuencial. Este valor es obtenido de la razón del tiempo de ejecución en un núcleo (secuencial) con respecto al tiempo de ejecución en múltiples núcleos. Para analizar este punto se observa la gráfica de la figura 7, de la cual se puede notar que el comportamiento real se aleja del ideal a medida que se incrementa el número de núcleos, pues el paralelismo tiene un límite. Este comportamiento se obtuvo al calcular

la aceleración de las ejecuciones del cálculo del producto matricial de dimensión 1200x1200 en el dispositivo móvil K011.

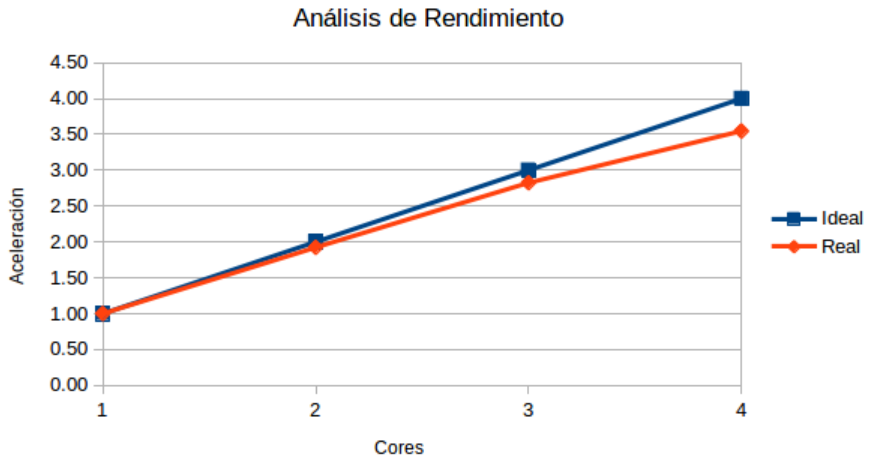

Figura 7. Análisis de rendimiento, aceleración.

Otra métrica presentada en la figura 8 como parte del análisis de rendimiento es la eficiencia, éste es el resultado de la razón de la aceleración y el número de núcleos. Observando que el valor de la eficiencia para cuatro núcleos es de 88.67% y de igual forma, decrece la eficiencia a medida que se incrementan los núcleos de acuerdo a la ley de Amdahl.

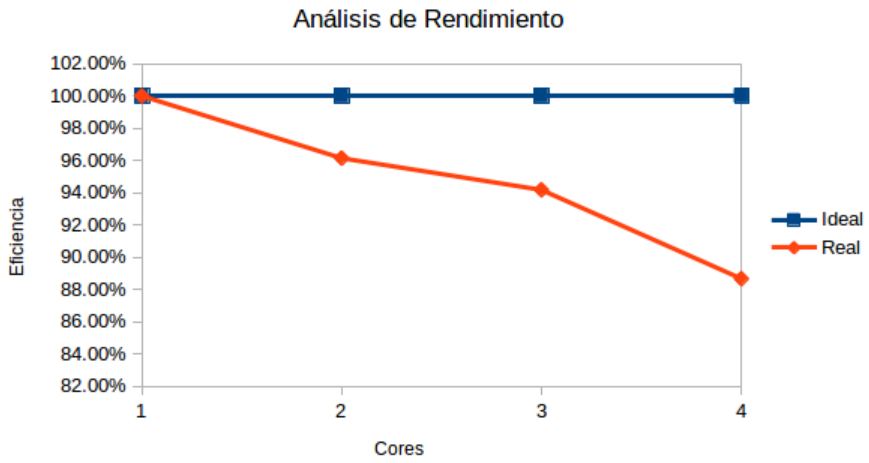

Figura 8. Análisis de rendimiento, eficiencia.

4. Conclusiones

El trabajo de investigación de aplicaciones móviles paralelas con OpenMP en arquitecturas multicore, nos muestra que se mejora el desempeño de las aplicaciones con respecto a las secuenciales.

Al desarrollar una aplicación móvil se debe seguir una metodología de programación en paralelo para su óptimo desempeño, aprovechando el hardware de las plataformas multicore.

Todo algoritmo tiene un límite paralelizable, por lo que se ha de realizar un análisis de rendimiento para identificar el costo de inversión tope del hardware hasta el punto en el cual ya no presenta un rendimiento significativo el incremento de núcleos en la arquitectura para ejecutar la aplicación.

Se propone como trabajo futuro el análisis del desempeño de aplicaciones paralelas en dispositivos móviles con procesadores multicore con otras tecnologías como CUDA (*Compute Unified Device Architecture*) y TBB (*Threading Bulding Blocks*), haciendo un análisis comparativo con respecto a OpenMP, así como la implementación de otros algoritmos paralelizables.

Bibliografía

[1] Amdahl, G. (2013, December). *Computer Architecture and Amdahl's Law. Computer*, 46, pp. 38-46.

[2] Ableson, F., Sen, R. & King, C. (2011). *Android: Guía para desarrolladores,* 2ª Edición. España: Anaya Multimedia.

[3] Basheer, D. & Al-Hafidh, M. (November, 2013). *Developing Parallel Application on Multi-core Mobile Phone*. International Journal of Advanced Computer Science and Applications, 4, pp. 89-93.

[4] Carroll, A. & Heiser, G. (June, 2010). *An analysis of power consumption in a smartphone*. in Proc. of USENIX. Boston, USA. p. 21.

[5] Chapman, B., Jost, G. & van der Pas, R. (2008). *Using OpenMP. Portable Shared Memory Parallel Programming*. USA: The MIT Press.

[6] D. an Mey. (2015). *OpenMP Application Program Interface*. agosto 15, 2015, de The Community of OpenMP Sitio web: http://www.compunity.org/

[7] Guihot, H. (2012). *Pro Android Apps Performance Optimization*. USA: Apress.

[8] Hill, M. & Marty, M. (July, 2008). *Amdahl's Law in the Multicore Era*. IEEE Computer Society, 08, pp. 33-38.

[9] Kegel, P., Schellmann, M. & Gorlatch, S. (2009). *Using OpenMP Vs. Threading Building Blocks for Medical Imaging on Multi-cores*. Euro-Par 2009 Parallel Processing - Computer Science. Eds. Springer. Berlin, Heidelberg. vol. 5704, pp. 654–665.

[10] OpenMP ARB. (2015). *The OpenMP API specification for parallel programming*. Agosto 15, 2015, de OpenMP Architecture Review Board Sitio web: http://openmp.org/wp/

[11] Reinders, J. (2007). *Intel Threading Building Blocks - outfitting C++ for multi-core processor parallelism*. USA: O'Reilly.

[12] Viso, E. (2013). *El reto de las arquitecturas multinúcleo.* Miscelánea Matemática, 56, 41-53.