Trabajo de Investigación: Integración de HPC en IOT

Adrian Radice and Facundo Sanabria

Universidad Nacional de La Matanza,
Departamento de ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
{radiceadriann,facundosanabria}@gmail.com
http://www.github.com/BiteselleSOA

Abstract. El objetivo del presente trabajo es investigar los beneficios de incorporar HPC en el proyecto de IOT *MatulaRubik* . Dicho proyecto tiene como desafío resolver un problema NP. Este problema es el puzzle Rubik 3x3x3, el cual se solucionara implementando el algoritmo no determinístico CFOP. CFOP varia de acuerdo cual se su cara de partida, por lo tanto para cualquier cubo se tienen 6 estados iniciales. El desafío es de 6 posibles caminos para llegar al estado final, quedarse con aquel que implique el menor uso del brazo lateral.

Keywords: HPC, IOT, Android, Rubik, Paralelismo, OpenCL

1 Introducción

MatulaRubik es un SE con el objetivo de resolver el Cubo Rubik de 3x3x3, en una primera instancia aplicando el algoritmo utilizado para competencias de velocidad CFOP, el cual obtiene un promedio entre 50 y 100 movimientos para resolver cualquier cubo. Este algoritmo pertenece a la familia de los llamados *Layer by Layer*, y se caracteriza por llevar a cualquier configuración del cubo a un estado particular, reduciendo siempre al problema a la misma solución. En pocas palabras mezcla aun mas el cubo para llevarlo a estados conocidos. CFOP es un algoritmo en donde un estado tienen distintas transiciones que nos llevan al estado final, siendo cada una finalmente un camino con distinta o igual cantidad de movimientos.

El objetivo de esta investigación, es reducir el tiempo en que *MatulaRubik* entrega la solución. *MatulaRubik* actualmente en su mecánica, tiene un retardo extra, la ubicación de las caras laterales en el correspondiente brazo para poder rotarlas. Se buscara mediante la implementación de HPC con OpenCL, encontrar la solución que reduce el retardo previamente descrito.

2 Desarrollo

Estudiando la mecánica de *MatulaRubik* se aprecia que el efectuar movimientos en las caras L, F, R, B pueden sumar un retardo, que no existe nunca al trabajarse

con las caras U y D. Por ejemplo la secuencia de movimientos F, B' implica, que luego de ejecutarse el primer movimiento, se sume un retardo temporal hasta que la cara B del cubo quede frente al brazo. Este retardo es el tiempo que implica retirar el brazo lateral, rotar el cubo hasta que B quede frente del brazo, y que el brazo lateral se ubique nuevamente en la posición de trabajo.

Enfrentar dicho problema implicaría dos caminos, cambiar la mecánica, o buscar una reducción en los movimientos de las caras problemáticas. En el desarrollo del trabajo optaremos por el segundo camino.

MatulaRubik en su estado actual aplica el algoritmo CFOP, el mismo presenta la particularidad que para una misma configuración del cubo de acuerdo a la cara que se tome como punto de partida devolverá una secuencia de movimientos totalmente distinto. Es decir, es como si se rotase lógicamente el cubo respecto de uno de sus ejes, lo que provoca que todas las piezas cambien de posición y orientación, resultando en un estado desordenado distinto y por lo tanto el algoritmo devolverá un conjunto de movimientos completamente distinto.

Para lograr obtener la menor cantidad de movimientos posibles utilizando este algoritmo nos proponemos a ejecutar en forma paralela 6 hilos de CFOP, un hilo por cara del cubo.

En busca de no tener que modificar demasiado el código actualmente implementado, se busco una herramienta que implique una implementación sencilla y optima de paralelismo. La herramienta seleccionada fue Paralldroid.

Paralldroid es un framework de desarrollo, que tiene como objetivo facilitar el trabajo de los desarrolladores de aplicaciones móviles ejecutadas sobre sistemas operativos Android, de acuerdo a lo mencionado en "Performance Analysis of Paralldroid Generated Programs" [1]. El mismo añade un conjunto de anotaciones al lenguaje Java, basadas en las definidas por el estándar OpenCL, permitiendo al desarrollador usar un lenguaje de alto nivel, sin entrar en detalle de implementación de paralelismo, y delegandole la responsabilidad a Paralldroid de generar el programa paralelo, segun lo indicado en "ParallDroid: A Framework for Parallelism in Android" [2]. Paralldroid puede ser descargado como plugin para eclipse, y solo es necesario instalarlo para que comience a funcionar[1].

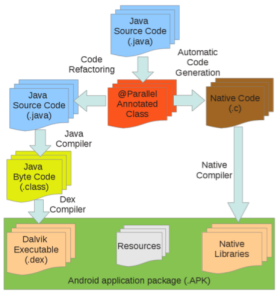

Fig. 1. Resultado de la refactorización [3].

Como se puede apreciar en la Figura 1 este framework, mediante el código Java con anotaciones, obtendrá un código nativo OpenCL, invocado desde nuestras clases. Es elemental el uso de NDK en cualquier App para poder hacer uso de este framework.

3 Distribución de operaciones

La particularidad de los hilos de trabajos propuestos, es que son independíentes entre si, cada uno trabaja con su propia matriz de estado del cubo, sin intervenir con los otros. Partiendo de que el resultado de ParallDroid es una implementación de OpenCL [1], cada hilo tendrá destinado un WorkGroup independíente. La primera ventaja de esta división, es que cada WorkGroup podrá ser ejecutado en una unidad de procesamiento distinta, si OpenCL lo considera [3], y como segunda ventaja es el escalado automático (asignación de WorkGroup a las unidades de procesamiento) que implementa OpenCL [3]. Esta ultima ventaja hace referencia, a que un mismo código, en un dispositivo de mayor cantidad de unidades de procesamiento que WorkGroups, en el mejor de los casos cada se asignara un WorkGroup por unidad de procesamiento, y en un dispositivo de menor cantidad, OpenCL simplemente terminaría ejecutando un conjunto de los WorkGroups en serie, en las unidades de procesamiento disponible.

4 Explicación Algoritmo

Nos basamos en el típico ejemplo de paralelismo del producto entre un escalar y un vector. Buscar 6 soluciones es isomórfico a esto ultimo, si entendemos al resultado de la operación como un conjunto de 6 elementos.

Pseudocódigo

```
@Parallel
public class SolverHPC_CFOP {
  private caras[];
  private  Solucion[];
 @Kernel
  private void Solver(CaraInicial){
         //CALCULAMOS LA SOLUCION
         Solucion [CaraInicial] <- CFOP(caraInicial);
  }
  private Solucion getBest(){
  //SE CALCULA LA MEJOR SOLUCION
  return Solucion[bestSol];
  }
}
public class Solver{
   SendSol(){
```

```
    SendArduino(SolverHPC_CFOP.getBest());
   }
}
```

Según lo mencionado en "Performance Analysis of Paralldroid Generated Programs" [1], ParallelDroid al refactorizar el código, agregaría el manejo de hilos, y nos abstrae de su desarrollo.

5 Pruebas que pueden realizarse

Para probar el rendimiento se propone, el desarrollo de un programa probador que mida el tiempo que le implica resolver 50 estados desordenados distintos del puzzle a a MatulaRubik con HPC y luego sin HPC. A partir de estos resultados debería realizarse el promedio para ambos casos y comparar cual obtiene el mejor tiempo.

También seria útil comprobar que el conjunto de movimientos resultados sean distintos dependiendo por que cara se comienza y si difieren mucho entre ellos.

6 Conclusión

La fortaleza de la solución propuesta en la investigación, es que permite una implementación de paralelismo sin demasiado esfuerzo de desarrollo. Se aprovecha el poder computacional provisto por el Hardware de los dispositivos móviles, para contrarrestar una deficiencia mecánica.

Extender la propuesta a cada uno de los distintos estados de CFOP en lugar de solamente los 6 iniciales seria computacionalmente intratable dado que CFOP es un algoritmo no determinístico, con una excesiva cantidad de estados posibles. Se compensaría los retrasos impuestos por la mecánica, con el tiempo de la búsqueda de mejor solución.

References

1. A. Acosta and F. Almeida, "Performance Analysis of Paralldroid Generated Programs" 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Torino, 2014, pp. 60-67.
2. Vicente Blanco, "ParallDroid: A Framework for Parallelism in Android".
3. Aaftab Munshi, "The OpenCL Specification" Khronos OpenCL Working Group, Revision 2015.