SmartFarm procesamiento de imágenes de plantas

Frattini, Maximiliano Gabriel
Rodeiro, Gonzalo
Salva, Ricardo Nicolás
Soro, Emmanuel

[1]Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina

Resumen. En el siguiente documento se explica la funcionalidad de obtener el ritmo de crecimiento de plantas mediante el procesamiento de imágenes en el proyecto SmartFarm. Dado que esto involucra una gran cantidad de imágenes, por tanto, un gran volumen de datos, emplearemos paralelismo con JCudaMP que es un entorno de desarrollo para el lenguaje de programación Java, involucra JaMP para proveer anotaciones estilo OpenMP para Java. Además, JCudaMP es un entorno capaz de reescribir el código para correr en hardware habilitado para CUDA en caso de detectarse dicho hardware, para así, aprovechar sus prestaciones. Con todo esto buscamos reducir los tiempos de procesamiento de los datos que comprenden las imágenes, aprovechando el paralelismo y sin tener un alto costo aparejado.

Palabras claves: HPC, Paralelismo, SmartFarm, JCudaMP, JaMP, OpenMP, CUDA.

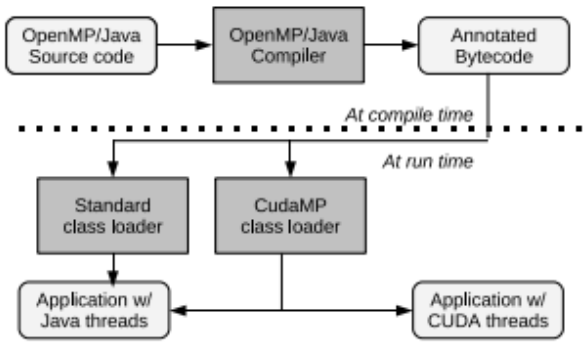
1 Introducción

SmartFarm es un proyecto IoT en el cual su función principal es monitorear el ritmo de crecimiento de las distintas plantas del usuario. Para cumplir esto, cuenta con sensores LDR (de luz) y luces leds para controlar el ambiente lumínico de la planta y emplea una cámara IP la cual toma las imágenes que son enviadas a un servidor web. Actualmente, debido a limitaciones, solo son tomadas 2 imágenes. En este documento abordaremos en la capacidad de poder obtener datos mas cercanos a la realidad sobre el crecimiento, y para esto requeriremos procesar una gran cantidad de imágenes.
El procesamiento digital de imágenes es una herramienta ampliamente utilizada en la automatización de procesos industriales. La industria agrícola ha comenzado a utilizar este tipo de tecnología para monitorear aspectos de gran relevancia en los cultivos. Para obtener datos confiables, se deben procesar múltiples imágenes de los cultivos, desde variados ángulos, lo que conlleva a un volumen de datos gigantesco.
Para poder ofrecer esta funcionalidad comenzaremos explicando como funciona el entorno de desarrollo JCudaMP y como es capaz de reescribir el código para que pueda correr en hardware habilitado para CUDA. Luego expondremos un pseudocodigo, explicaremos las pruebas que pueden realizarse y, por último, daremos una conclusión.

2 Desarrollo

JCudaMP

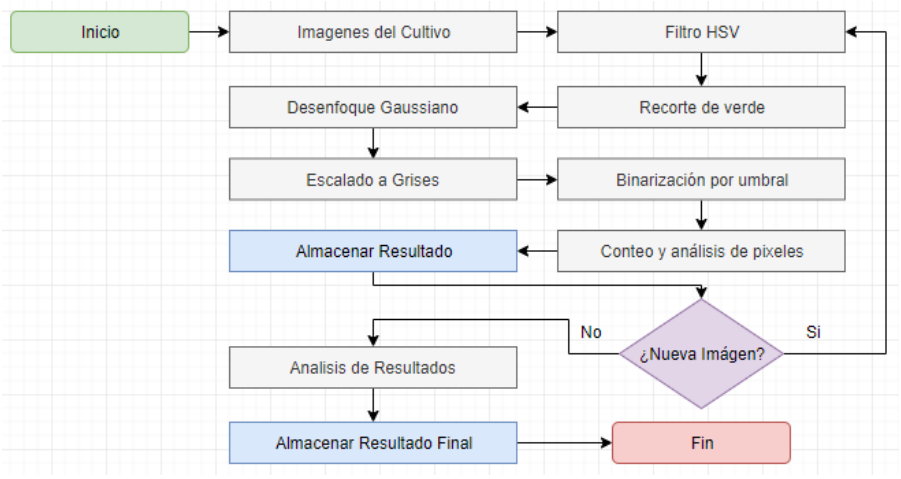
Este framework fue explicado en el paper *JCudaMP: OpenMP/Java on CUDA*[1]. Es un modelo capaz de analizar dinamicamente el entorno de ejecucion para determinar el hardware disponible. Dependiendo del hardware, se reescribira el bytecode y generará el necesario para que pueda correr en gpGPU. JCudaMP utiliza JaMP, que proporciona anotaciones tipo OpenMP en el codigo Java. Esto es necesario para eludir la programacion paralela de bajo nivel de Java y centrarnos en el paralelismo de estilo SPDM que es adecuado para GPU. El compilador JaMP es un compilador de OpenMP que traduce el codigo OpenMP/Java a bytecode Java multihilos. Este compilador es extendido para poder emitir anotaciones en ese bytecode Java, tendremos una nueva clase llamada Java Class Loader que reconocera estas anotaciones y estará encargada de realizar la reescritura del bytecode para funcionar en el hardware detectado. A continuación, tenemos una imágen donde se puede ver el modelo de ejecución de un programa OpenMP/Java en un entorno CUDA.



Algoritmo de Imagen

Para desarrollar el algoritmo nos basamos en el paper *Procesamiento de Imágenes de plantas ornamentales multiescala para calcular su procesamiento*[2] y en el paper *Enseñanza del Procesamiento de Imágenes en Ingeniería usando Python*[3]. En el documento se explican las dificultades que conlleva el procesamiento de imágenes de los cultivos, los diversos factores que provocan que las imágenes difieran en el tiempo, no solo por el crecimiento del cultivo sino también por factores climáticos, como el viento. Por esto, se da la necesidad de tomar una gran cantidad imágenes desde múltiples ángulos (al menos 3) y luego de procesarlas, cruzar los datos obtenidos para poder llegar a poder establecer de manera correcta el ritmo de crecimiento del cultivo. Este procesamiento será realizado en un web service de la aplicación, que entregará los datos finales a la aplicación de Android, donde el usuario podrá ver el ritmo de crecimiento histórico de los cultivos mediante gráficos.

De esta manera, dejamos fuera de la aplicación Android el arduo trabajo de almacenar las imágenes y procesarlas. A continuación, tenemos una imágen que muestra al algoritmo de procesamiento de imágenes.



Queda claro que las imágenes pueden ser procesadas de manera paralela, ya que el procesamiento de cada una es independiente de el de las demás.

3 Explicación del algoritmo.

A continuación, se detalla un pseudocódigo de cómo se podría implementar, de manera global, los métodos anteriormente mencionados. Se introduce en el código Java nuevos anotaciones OpenMP.

```
public static void main{

        Global List<Image> imglist = new List<>(); //Lista que tendrá las imágenes

        new double data; //Almacenara el resultado final

        capturarImagenes(); //Hilo que agrega imagenes recibidas de la camara IP

        Global double img_vec_data[] = new double[imglist.size]();

        //#pragma omp parallel for nowait

        for(int i=0; i<imglist.size; i++){

//Procesa la imagen en la pos i de la lista y guarda el resultado en la pos i de img_vec_data
```

```
            procesarImagen(i);
        }
//Cruza los datos que se encuentran en img_vec_data y devuelve un único resultado que representara el nivel de follaje actual del cultivo.
        data = cruzarDatos();
//Almacena el dato en la BD para posteriormente poder realizar graficos historicos
        guardarData(data);
}
public void procesarImagen(pos) {
        int cant_pix_blanco = 0; //Variables
        int cant_pix_tot = CANT_PIX_FIL * CANT_PIX_COL;
        Image img = imglist.img; //Tomo una imágen de la lista
        imglist.delete(imglist.IndexOf(img)); //Elimino la imagen de la lista
        //#pragma parallel for collapse(2)
        for(t=0; t<CANT_PIX_FIL;t++)
                for(k=0;k<CANT_PIX_COL;k++)
                        img[t][k].HSV() //Pasamos la imagen de RGB a HSV
        //#pragma parallel for collapse(2)
        for(t=0; t<CANT_PIX_FIL;t++)
                for(k=0;k<CANT_PIX_COL;k++)
                        if(img[t][k].noVerde())
                                img[t][k]=0; //Si no es verde, lo ponemos negro.
        img.reducirRuidoPorGauss(); //Elimino el ruido de la imagen, no se puede paralelizar.
```

```
        //#pragma parallel for collapse(2)
        for(i=0;i<CANT_PIX_FIL;i++)
                for(j=0;j<CANT_PIX_FIL;j++){
                        img[i][j].Agris();
                        img[i][j].Binarizar();
                        if(img[i][j].Blanco())
                                cant_pix_blanco++;
                }
        img_vec_data[pos] = (cant_pix_blanco * 100) / cant_pix_tot;
}
```

4 Pruebas que pueden realizarse

Los casos de prueba que se pueden realizar son variados. Un primer caso seria correr el algoritmo en sistemas de diferentes características. Por ejemplo, en un Sistema monoprocesador, en uno multiprocesador, y en un sistema que disponga de hardware habilitado para CUDA. Así, podremos evidenciar de manera clara los distintos tiempos de ejecución para cada arquitectura.
Luego podríamos evidenciar como se comporta el algoritmo dependiendo de la cantidad de la entrada que disponga en cada sistema mencionado anteriormente.

5 Conclusiones

El presente trabajo da una breve introducción de cómo se podría aprovechar distintas tecnologías de procesamiento paralelo para integrar a nuestro presente proyecto. Estas tecnologías (GPU, OpenMP, JCudaMP) son optimas ya que pueden funcionar en un entorno de trabajo con limitaciones, no requieren un costoso hardware como un supercomputador. Por esto, se propuso una alternativa para procesar de manera paralela grandes volúmenes de datos en un pequeño servidor.
Presentamos el entorno de desarrollo JCudaMP, que esta al alcance de cualquier desarrollador que quiera aventurarse en el mundo de la GPGPU. Este framework es relativamente nuevo, pero el uso de las GPU como GPGPU ya es algo que se da hace unos años. Así, se puede obtener un gran beneficio de su poder de procesamiento altamente paralelo. Por tanto, notamos que explotar este tipo de herramientas provee un gran beneficio.

6 Referencias

1. Georg Dotzler, Ronald Veldema, Michael Klemm, University of Erlangen-Nuremberg – Computer Science Department 2, Germany: JCudaMP: OpenMP/Java on CUDA.
2. Césares Farías, C.A., Farías Mendoza, N., García Díaz, N. y García Rebolledo, A. (2017). Procesamiento de imágenes de plantas ornamentales multi-escala para calcular su crecimiento. 3C TIC: Cuadernos de desarrollo aplicados a las TIC, 6(3), 10-25.
3. Andres Fernando Jimenez Lopez, Marla Carolina Prieto Pelayo, Ángela Ramírez Forero: Enseñanza del Procesamiento de Imágenes en Ingeniería usando Python. VAEP-RITA Vol. 3, Núm 4, dic. 2015.
4. Petr B˘elohl´avek: OpenMP for Java. Department of Distributed and Dependable Systems. Prague 2015.
5. José Antonio Taquía Guitiérrez: El procesamiento de imágenes y su potencial aplicación en empresas con estrategia digital. Universidad de Lima, Colombia. 15 de Octubre del 2017.