Trabajo de Fin de Máster

Máster en Ingeniería Informática

Servicio web de GPU

Autor
José Cristóbal López Zafra

Tutor
Maria Isabel García Arenas

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 29 de agosto de 2016

Servicio web de GPU

José Cristóbal López Zafra

S W G P U

Figura 1: SWGPU

Índice general

Índice de figuras

Capítulo 1

Resumen

1.1. Resumen y palabras clave

Palabras clave: *servicio web*, *C++*, *GPU*,*evaluación de algoritmos*, *CUDA*.

El objetivo principal es el desarrollo de un conjunto de clases en C++ que permitan que un servicio web interactúe con ellas. Esa infraestructura que da soporte a las llamadas del servicio debe hacer uso de una GPU, usando la arquitectura de cálculo CUDA.

Para ello implementaremos una infraestructura necesaria para crear servicios Web y la ampliaremos para crear los servicios web basados en GPUs: dicho servicio llamará a la GPU para lanzar un algoritmo evolutivo, y usar una función de evaluación mediante un algoritmo (o varios) por determinar.

1.2. Extended abstract and key words

Key words: *web service*, *C++*, *GPU*, *evaluation algorithms*, *CUDA*.

The main objective of this project is ...

Capítulo 2

Introducción, motivación y objetivos

2.1. Introducción

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico. A principios de la década de 1960, en 1962 John Henry Holland ideó una de las líneas más prometedoras de la inteligencia artificial, la de los algoritmos genéticos, y con su libro .Adaptation in Natural and Artificial Systems.en 1975 logró una mayor popularidad.

Son llamados así porque se inspiran en la evolución biológica y su base genético-molecular. Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados.

Para su funcionamiento (el de un algoritmo genético simple, llamado Canónico), al empezar necesita una codificación o representación del problema que se adecue al mismo y una función de ajuste o adaptación al problema. Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción, a continuación dichos padres seleccionados se cruzarán generando dos hijos, sobre cada uno de los cuales actuará un operador de mutación. El resultado de la combinación de las anteriores funciones será un conjunto de individuos (posibles soluciones al problema), los cuales en la evolución del algoritmo genético formarán parte de la siguiente población.

Poseen multitud de aplicaciones, como pueden ser:

- Diseño automatizado para la investigación de diseño de materiales, equipamiento industrial o sistemas de comercio en el sector financiero.
- Construcción de árboles filogenéticos.
- Diseño de sistemas de distribución de aguas.
- Resolución de equilibrios en Teoría de juegos.
- Análisis de expresión de genes.

Pueden resolver problemas de alta complejidad, pero esto suele traducirse en operaciones muy costosas en términos de tiempo y recursos. En la actualidad, por ejemplo, hay casos en los que recrear una simulación de la solución propuesta por una iteración puede durar varios días y consumir gran cantidad de procesamiento y recursos asociados.

2.2. Motivación

Dicha complejidad en la resolución de los algoritmos genéticos nos lleva a optimizarlos y a trabajar con ellos para lograr reducir el coste de dichas operaciones.

Esto nos lleva a utilizar una computación paralela, en la que se aprovecha la potencia de computación de los sistemas con los que trabajamos realizando cálculos simultáneamente. Sigue el principio de dividir los problemas grandes en problemas más pequeños, que se solucionan en paralelo para luego obtener la solución del problema inicial.

Por otra parte, en el ámbito de la computación se ha visto una gran evolución de las tarjetas gráficas, ya que grandes fabricantes como NVIDIA, AMD, IBM o Intel han logrado una producción de GPUs de gran alcance, como se puede ver en la Figura 2.1 (pueden incluso superar la frecuencia de reloj de una CPU antigua).

El uso de computación paralela, junto la potencia y la posibilidad de usar el paralelismo que ofrecen las GPU hace muy interesante el uso de dichas GPUs para resolver algoritmos genéticos de formas más eficiente.

Después del estudio realizado en el siguiente capítulo, se opta por usar la arquitectura de cálculo CUDA [1] para crear algoritmos genéticos que se ejecuten en GPUs de NVIDIA [2].

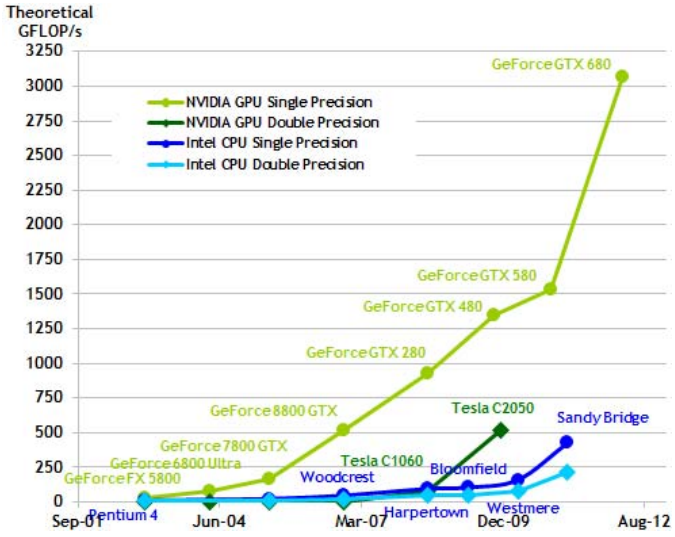

Figura 2.1: Evolución hasta 2012 de las GPUs respecto a las CPUs

2.3. Objetivos

Se desarrollarán varias clases para lanzar algoritmos genéticos que aprovechen el cálculo paralelo que ofrece CUDA.

Además se desarrollará un servicio web que interactúe con ellas. Esa infraestructura que da soporte a las llamadas del servicio debe hacer uso de una GPU. Esto hará que dichos algoritmos genéticos sean más accesibles y fáciles de usar, ya que tendrán que introducir los parámetros que se deseen y lanzar el algoritmo de manera intuitiva y sin necesidad de una preparación previa.

Analizaremos la respuesta del algoritmo que utiliza los servicios basados en GPUs respecto a los algoritmos que no la utilizan.

Y se estudiará la posibilidad de publicar dicho servicio para que sea accesible por cualquier usuario de Internet, dependiendo del rendimiento obtenido.

Capítulo 3

Estudio del arte

Son muchos y muy variados los ámbitos en los que se usan los algoritmos genéticos. Sirven para crear componentes automovilísticos, analizar expresiones de genes o hasta desarrollar aprendizajes de comportamiento para robots.

Esto hace que se estudie y se intente mejorar y optimizar dichos algoritmos, siendo un campo con mucha transcendencia en la actualidad.

Aún así, en los primeros años de los algoritmos genéticos, se estudiaron y se lograron las ideas básicas que ahora se intentan mejorar. Constan de varias partes, como la selección de candidatos, operadores de cruce, funciones de evaluación y optimización o mutación.

En este trabajo nos centraremos en la parte de optimización, donde se evalúan los individuos generados y que serán posibles soluciones al problema.

En 1987 Ackley realiza un trabajo donde añade una función de optimización. [completar !!!]

Años antes, en 1974, Rastringin en "Systems of extremal control."propone otra función de optimización, para ser generalizada en 1991 por Mühlenbein. [completar !!!]

En este trabajo nos centraremos en estas 2 funciones.

3.1. Actualidad

En la actualidad la mayoría de los usuarios que necesitan procesar un algoritmo genético lo hacen ellos mismos, con programas adaptados de terceros o desarrollados por ellos mismos. Esto conlleva un trabajo extra de estudio, implementación y corrección de dichos programas.

Podemos encontrar algunos programas [3] [4] [5] para ser ejecutados por el cliente, o servicios web que realizan algoritmos genéticos donde la mayoría son ejemplos o demostraciones de algoritmos genéticos y sus soluciones [6] [7].

Pero ninguno usa la computación paralela aprovechando la potencia de las GPUs. Viendo que hay disponible en el ámbito de dicha paralelización vemos que hay grandes empresas que han lanzado lenguajes de programación enfocados a aprovechar el procesamiento mediante la GPU: CUDA [1], AMD OpenCL APP [8], BrookGPU [9], PeakStream o RapidMind:

- CUDA: arquitectura de cálculo paralelo de NVIDIA. Se basa en el lenguaje C y C++, por lo que, junto con la cantidad de dispositivos de la marca, existe una gran comunidad y documentación.
- AMD OpenCL™ Accelerated Parallel Processing: herramienta de AMD que permite el cómputo mediante GPUs. Se basa en los lenguajes OpenCL y C++, por lo que se pueden usar para acelarar aplicaciones.
- BrookGPU: programa (en versión *beta*) de la Universidad de Stanford para aprovechar la paralelización en tarjetas gráficas AMD y NVIDIA. Para trabajar con el se usa una extensión de ANSI C.
- PeakStream era un programa para paralelizar el procesamiento con grandes rendimientos en tarjetas AMD, comprado por Google en 2007 [10], y tras esto, a dejado de ofrecer mantenimiento.
- RapidMind, que tambíen se basa en C++, es comprada por Intel en 2009 y pasa a ser Intel Array Building Blocks [11], pero sigue estando en forma experimental.

Tras ver las distintas opciones, con sus distintas ventajas e inconvenientes, decidimos centrarnos en CUDA. Vemos que es un proyecto activo, tiene numerosas actualizaciones, cuenta con mucha comunidad para resolver dudas y fomentar su desarrollo, y tiene numerosas facilidades, como un SDK [2] y varias herramientas para sus desarrolladores. Por todo esto escogemos CUDA para desarrollar nuestro trabajo.

En este campo hay algunos proveedores, como Impact [12] o gpuOcelot [13] que ofrecen herramientas para la computación de algoritmos estándar o del código que genere el usuario, pero realizando el cómputo desde las

GPUs de los clientes, y nunca enfocados específicamente a resolver algoritmos genéticos.

Otras opciones nos permiten ejecutar CUDA en servidores externos, pero se necesita enviar una solicitud de uso y adaptarse a sus restricciones y capacidad, como en rCUDA [14] o desplegar toda una instancia en un IaaS y desarrollar dentro: Amazon Web Services [15].

Con esto vemos que podemos encontrar servicios que realicen algoritmos genéticos, o servicios que usen CUDA, pero no hay servicios que combinen ambos.

3.2. Clientes

¿Que usuarios necesitan computar algoritmos genéticos? Por ejemplo, cualquier estudiante que quiera comprobar los cambios en los distintos parámetros del algoritmo genético.

¿Y además algoritmos que requieran mucha capacidad de cómputo? En principio el personal de investigación cumple con estas características. Junto con la potencia de cómputo y la accesibilidad que proporcionaremos simplificaríamos el trabajo de dicho personal.

Además el servicio será accesible a cualquier usuario, y con su interfaz sencilla e intuitiva dichos usuarios no necesitarán una preparación especial para usarlo.

3.3. Competidores

Como se cita antes, existen ejemplos o plantillas de algoritmos genéticos [3] [4] [5] que los usuarios pueden usar, pero necesitan para su uso un trabajo extra para su instalación, desarrollo o ajustes. Además de este trabajo extra, se ven limitados por las capacidades de sus dispositivos, pues los algoritmos se tendrán que lanzar en local.

Para evitar dichas limitaciones, podemos usar la parelelización de los algoritmos, pero los trabajos que encontramos se encuentran en una versión de CUDA obsoleta [16] (y simplemente exponiendo el código) o son estudios del servicio sin llegar a ser implementado [17]. Se busca entonces un servicio que ofrezca computación desde un servidor externo. Podemos ver servicios que computan directamente CUDA [14] u ofrecen instancias con acceso a GPUs [15] pero nunca especializados en algoritmos genéticos.

3.4. Conclusiones

Si buscamos un servicio de computación externo (que ofrezca una mayor potencia de computación usando GPUs) de algoritmos genéticos no llegamos a encontrar nada que cumpla con nuestro requisitos: o son servicios en local para lanzar algoritmos genéticos o son servicios externos que nos ofrecen computación aprovechando la paralelización de CUDA, pero sin llegar a ofrecer ninguna aproximación de un algoritmo genético.

Viendo esto llegamos a una demanda que podemos cubrir con nuestro servicio, motivando a su desarrollo, implementación y creación.

Capítulo 4

Resolución del trabajo

Enumeraremos y describiremos las herramientas y tecnologías de desarrollo que se han usado.

4.1. Planificación

4.2. Tecnologías

4.2.1. Servidor

4.2.2. Cliente

4.3. Herramientas

Capítulo 5

Descripción del sistema

Detallaremos las restricciones que se han tenido al empezar el desarrollo, los requisitos funcionales identificados, y los casos de uso.

5.1. Análisis del sistema

5.2. Objetivos

5.3. Resumen de implicados

5.4. Especificación de requisitos

5.4.1. Requisitos funcionales

5.4.2. Requisitos no funcionales

5.4.3. Requisitos de información

5.4.4. Restricciones

5.5. Diagramas del sistema

5.5.1. Diagramas de clases

5.5.2. Diagramas de caso de uso

Capítulo 6

Desarrollo del sistema

En este capítulo veremos la arquitectura del sistema, la *filosofía* o principios seguidos y algunos detalles de la implementación.

6.1. Arquitectura del sistema

6.2. Partes del sistema

6.3. Filosofía a seguir

A la hora de implementar y de realizar el trabajo en general se ha intentado ser lo más ordenado y pulcro posible, esto en un principio puede ralentizar el proceso, pero a largo plazo hace que el desarrollo, actualización o mantenimiento del sistema sea más rápido y fácil. Estos son algunos de los criterios empleados en el desarrollo.

6.3.1. Desarrollo general

Las funciones del sistema se han desarrollado de la manera más general posible para así favorecer la reutilización de código y facilitar su legibilidad. También son más fáciles de mantener puesto que al ser de ámbito más general son más mantenibles.

6.3.2. Modularización

Se ha desarrollado el código en distintos módulos. De esta forma se evita que al hacer cambios en un módulo se propague a los demás, lo que hace el código más mantenible y eficiente.

6.3.3. Control de versiones

Se apostó por un sistema de control de versiones, en mi caso Git mediante la plataforma GitHub, ya que favorece el mantenimiento de las distintas versiones de código de una manera sencilla y rápida.

El manejo de Git es sencillo, ya que con unas cuantas ordenes se puede manejar sin problemas, y GitHub está provisto de una interfaz muy intuitiva. Algunas de las órdenes iniciales para su manejo son:

git clone[url del proyecto]: descargamos el proyecto del repositorio Git.

git add [archivos]: añade los archivos con los cambios deseados en un "paquete"para el commit.

git commit: subida de los archivos especificados al repositorio local, a diferencia de otros sistemas como SVN con el commit no se propaga el cambio.

git push: propaga los cambios locales al repositorio.

git pull origin: Actualiza la versión del código.

git status: muestra el estado de los archivos.

6.3.4. Desarrollo iterativo incremental

El desarrollo de las funcionalidades se hace de forma progresiva, de modo que primero se implementan las más críticas y prioritarias en primer lugar para poder tener siempre un producto funcional.

6.3.5. Revisiones periódicas del código

El código es revisado tras cada interacción de desarrollo con el fin de mantenerlo lo más pulcro y estructurado posible. De este modo evitamos repeticiones en el código o dejar partes incompletas. Así proporcionamos un mayor nivel de calidad a código producido.

6.3.6. Documentación del código

El código a sido documentado, de modo que es más mantenible por terceros y por el mismo autor. Cada funcionalidad ha sido detallada debidamente, de manera clara y concisa, sin extender demasiado la explicación.

6.4. Codificación y estructuración

6.4.1. FrontEnd

6.4.2. BackEnd

6.4.3. Ejecutables a usar por el BackEnd

6.5. Pruebas

Capítulo 7

Manual de usuario

...

Capítulo 8

Conclusiones y trabajos futuros

..

8.1. Conclusiones

8.2. Trabajos futuros

Bibliografía

[1] "Procesamiento paralelo mediante cuda."
http://www.nvidia.es/object/cuda-parallel-computing-es.html.

[2] "Nvidia developer."
https://developer.nvidia.com/. (Accessed on 01/07/2016).

[3] "Algoritmo genético en python."
http://robologs.net/2015/09/01/como-programar-un-algoritmo-genetico-parte-ii-implem

[4] "Jgap: Framework java para realizar algoritmos genéticos."
http://jgap.sourceforge.net/.

[5] "'global optimization toolbox' para matlab."
http://es.mathworks.com/products/global-optimization/.

[6] "Simulación de algoritmo genético del aprendizaje de un robot a andar."
http://rednuht.org/genetic_walkers/.

[7] "Simulación de algoritmo genético en la creación de la forma de un vehículo."
http://gencar.co/.

[8] "Amd opencl^{TM} accelerated parallel processing."
http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/.

[9] "Brookgpu."
http://graphics.stanford.edu/projects/brookgpu/.

[10] "Peackstream comprado por google."
https://www.crunchbase.com/organization/peakstream#/entity.

[11] "Rapidmind es comprada por intel."
https://software.intel.com/en-us/articles/intel-array-building-blocks.

[12] "Impact: Cuda framework."
http://impact.crhc.illinois.edu/mcuda.aspx.

[13] "gpuocelot: Framework dinámico de compilación que soporta cuda."
https://code.google.com/archive/p/gpuocelot/.

[14] "Framework rcuda para ejecutar programas cuda en un servidor externo."
http://www.rcuda.net/index.php/what-s-rcuda.html.

[15] "Instancia en amazon web services que permita usar la gpu."
https://aws.amazon.com/blogs/aws/new-ec2-instance-type-the-cluster-gpu-inst

[16] "Trabajo sobre la paralelización de algoritmos basados en poblaciones por medio de gpgpu."
http://posgrado.itlp.edu.mx/uploads/4f335b50b576b.pdf.

[17] "Optimización en paralelo basada en poblaciones usando gpgpu."
https://dialnet.unirioja.es/servlet/articulo?codigo=3985066.