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, 4 de septiembre 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*.

En este trabajo se aborda la creación de un conjunto de clases que permitan lanzar algoritmos genéticos y ser optimizados. Para un máximo rendimiento se paralelizarán los algoritmos, haciendo uso de una GPU.

Acceder a esas funcionalidades será posible desde un servicio web que interactue con dichas clases.

Para ello implementaremos una infraestructura de un servicios web y la ampliaremos a un servicio web que haga uso de GPUs: dicho servicio llamará a la GPU para lanzar un algoritmo evolutivo, y usar una función de evaluación mediante el algoritmo de Ackley o Rastrigin.

Esa infraestructura que da soporte a las llamadas del servicio usará la arquitectura de cálculo CUDA.

Al finalizar el trabajo obtenemos un servicio web que permite lanzar algoritmos genéticos, haciendo uso de la GPU de un servidor externo mediante CUDA.

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

Objetivos y motivación

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 Artifical 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.

2.1. Objetivos

- Se desarrollarán varias clases para lanzar algoritmos genéticos que aprovechen el cálculo paralelo de alguna arquitectura de cálculo especializada (más adelante se vera que se opta por 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 sólo tendrán que introducir los parámetros que se deseen y lanzar el algoritmo de manera intuitiva y sin necesidad de una preparación previa.
- Y se estudiará la posibilidad de publicar dicho servicio para que sea accesible por cualquier usuario de Internet, dependiendo del rendimiento obtenido.

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).

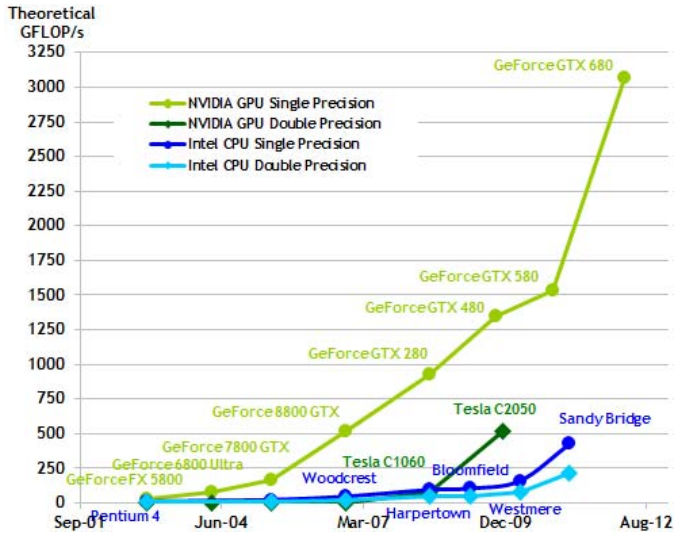

Figura 2.1: Evolución hasta 2012 de las GPUs respecto a las CPUs

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 [31] para crear algoritmos genéticos que se ejecuten en GPUs de NVIDIA [28].

Como se cita en uno de los objetivos, se publicará de manera que cualquier usuario tenga acceso. Esto, junto a una interfaz sencilla, permitirá computar algoritmos genéticos con facilidad y rapidez.

Capítulo 3

Estado del arte

Los principios básicos de los algoritmos genéticos fueron establecidos por Holland (1975), y se encuentran bien descritos en varios textos a lo largo de los años, como los estudios de Goldberg (1989), Davis (1991), Michalewicz (1992) o Reeves (1993), afianzando los conceptos y permitiendo trabajar sobre una buena base a futuros investigadores.

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.

El poder de los algoritmos genéticos proviene del hecho de que se trata de una técnica robusta y tratan con éxito una gran variedad de problemas provenientes de diferentes ámbitos, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el algoritmo genético encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de los algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que se superen al Algoritmo genético, tanto en rapidez como eficacia. El gran campo de aplicación de los algoritmos genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas conjuntándolas con los algoritmos genéticos.

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

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.

Existen multitud de funciones de optimización, pero nos centraremos en 2: Ackley y Rastrigin.

La función de Ackley se publicó por primera vez en .ª connectionist machine for genetic hillclimbing"por David H. Ackley en 1987 y se extendió a la dimensión arbitraria en .ªvolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms"por Thomas Bäck en 1996.

$$f(x) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$$

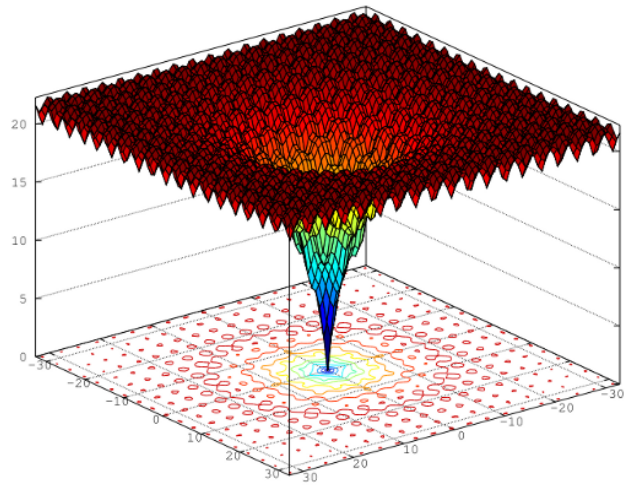

Figura 3.1: Función y representación de la función de optimización de Ackley

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

En este trabajo nos centraremos en estas 2 funciones de optimización, ofreciendo un algoritmo genético que pueda ser optimizado mediante Ackley o Rastrigin.

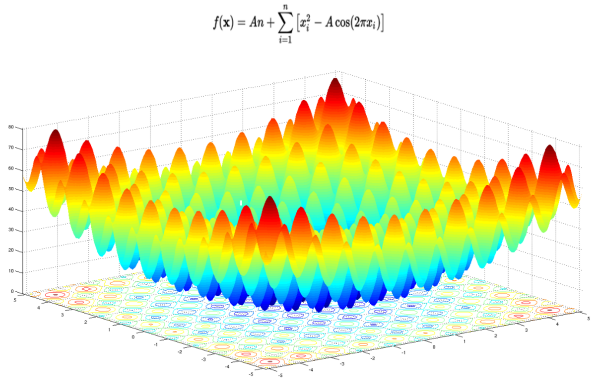

Figura 3.2: Función y representación de la función de optimización de Rastrigin

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 [1] [23] [16] 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 [34] [35].

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 [31], AMD OpenCL APP [4], BrookGPU [7], 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 [30], 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 [33], 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 [28] y varias herramientas para sus desarrolladores. Por todo esto escogemos CUDA para desarrollar nuestro trabajo.

CUDA surgió en 2006, cuando NVIDIA lanzó por primera vez al mercado una GPU capaz de renderizar gráficos en 3D y que incluyó la posibilidad de ejecutar, usando CUDA, programas escritos en lenguaje C.

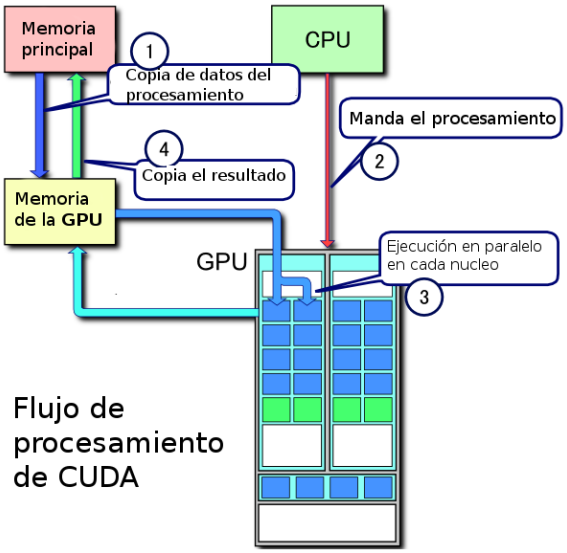

Figura 3.3: Ejemplo de flujo de procesamiento CUDA

Gracias a su eficiencia energética y gran potencia, desde entonces NVIDIA ofrece la capacidad de sus tarjetas gráficas para grandes centros de datos y otras instituciones tanto científicas como gubernamentales, permitiendo dibujar gráficos y ejecutar diversos programas aprovechando.

Incluso se utilizan versiones reducidas para potenciar las aplicaciones tanto de teléfonos móviles, ordenadores portátiles y tablets entre otros.

En este campo hay algunos proveedores, como Impact [20] o gpuOcelot [17] 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 [12] o desplegar toda una instancia en un IaaS y desarrollar dentro: Amazon Web Services [21].

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 [1] [23] [16] 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 paralelización de los algoritmos, pero los trabajos que encontramos se encuentran en una versión de CUDA obsoleta [38] (y simplemente exponiendo el código) o son estudios del servicio sin llegar a ser implementado [29]. Se busca entonces un servicio que ofrezca computación desde un servidor externo. Podemos ver servicios

que computan directamente CUDA [12] u ofrecen instancias con acceso a GPUs [21] 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

La planificación del proyecto se realizó al inicio del mismo y luego se fue alterando según las desviaciones que se producían. El inicio del proyecto fue el 1 de Abril, aunque hasta el 1 de Julio se trabajó en él de manera entrecortada y sin profundidad, y la fecha de fin planificada es el 10 de Septiembre.

Se podrán distinguir varias etapas en el trabajo, que se ven mejor en la siguiente imagen (Figura 4.1) de la planificación:

- Preparación y análisis: Aquí se hizo un análisis inicial del proyecto, con tecnologías a usar y un estudio más profundo de algunos conceptos vistos en el curso que se abordarían.
- Desarrollo del algoritmo genético: esta etapa consta de varias partes, ya que primero se trabajó con distintos algoritmos simples, genéticos o que usaban la paralelización a forma de formación y para facilitar un posterior diseño y análisis del algoritmo que se implementará. A medida que se acababa la implementación, se hicieron pruebas con los resultados de ambas funciones de optimización.
- Desarrollo del servicio web: también consta de varias etapas. Primero se creo el BackEnd, y como el FrontEnd requería de los resultados del algoritmo, no se desarrolló hasta obtener algún resultado de este. Una vez completadas estas partes, se pasa al despliegue con una versión estable.
- Documentación: Aunque se ha ido preparando y almacenando información y recursos durante todo el trascurso del trabajo, la mayor parte de la memoria del trabajo se crea y se plasma en este documento en la etapa final, haciendo las correcciones y mejoras pertinentes.

4.1. Planificación

A continuación se muestra mediante un diagrama de Gantt la planificación estimada del trabajo.

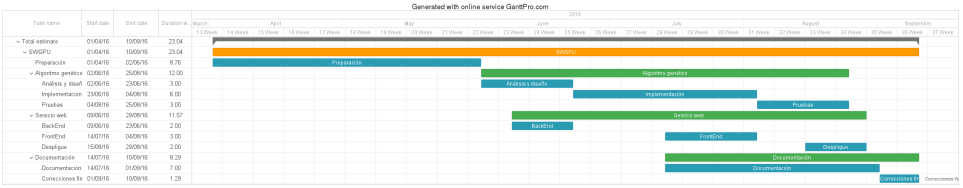

Figura 4.1: Diagrama Gantt de la planificación prevista del trabajo

Y en la siguiente figura se ve el desarrollo que se a producido, tras desviaciones imprevistas o ajustes de tiempo durante el mismo proceso:
(aproximación del diagrama de Gantt, pendiente la definitiva:)

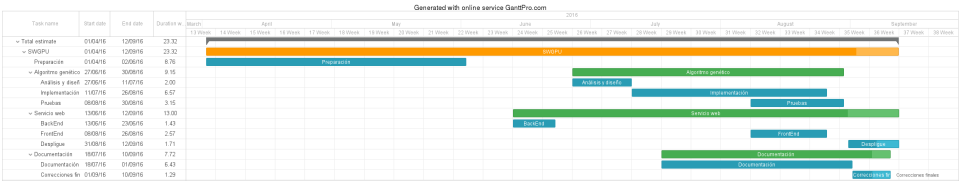

Figura 4.2: Diagrama Gantt de la planificación del trabajo

4.1.1. Conteo de horas y presupuesto

Conteo de horas

- Estimado
  - Preparación:
    8.75 semanas, a 3 días/semana, a 2 horas/día: 52.5 horas
  - Desarrollo de algoritmo genético:
    12 semanas, a 4 días/semana, a 4 horas/día: 192 horas
  - Desarrollo de servicio web:
    7 semana, a 5 días/semana, a 2 horas/día: 70 horas
  - Documentación:
    8.29 semanas, a 4 días/semana, a 2 horas/día: 66.3 horas

  Con lo que se sumaría un total de 380.8 horas.

- Real
  - Preparación:
    8.76 semanas, a 3 días/semana, a 2 horas/día: 52.5 horas
  - Desarrollo de algoritmo genético:
    9.15 semanas, a 4 días/semana, a 4 horas/día: 146.4 horas
  - Desarrollo de servicio web:
    ?? semanas, a 5 días/semana, a 2 horas/día: ?? horas
  - Documentación:
    ?? semanas, a 4 días/semana, a 2 horas/día: ?? horas

    Sumando finalmente un total de ?? horas.

Presupuesto

En este punto veremos los presupuesto según el conteo de horas para cada una de las 2 estimaciones y los servicios que requiramos. Suponiendo la media actual de un programador Junior [26]: 9€/hora

- Estimado
  - Preparación:
    52.5 horas x 9€/hora : 472,5€
  - Desarrollo de algoritmo genético:
    192 horas x 9€/hora: 1728€
  - Desarrollo de servicio web
    70 horas x 9€/hora: 630€
  - Documentación:
    66.3 horas x 9€/hora: 596,7€
  - *Instancia que permita usar CUDA*
    Se contratarán servicios externos de cómputo mediante CUDA [21] con un coste de 1.88€/h (2.10$/h). Suponiendo que este servicio estuviese en total disponibilidad (24 horas los 7 días de la semana) alcanzaría en sólo un mes de servicio los 315,84€.
  - *Dominio para alojar una web publica del servicio*
    En caso de que no fuese posible, se contrataría un dominio por precios de 2.08€a 4.95€al mes (con permanencia de 12 meses, en las webs de alojamiento *Hostpapa.es* [3] y *Dondominio.com* [2])

    Presupuesto total estimado: 3745,12€(para el desarrollo y sólo un mes en producción)

- Real
  - Preparación
    52.5 horas 9€/hora
  - Desarrollo de algoritmo genético
    146.4 horas 9€/hora
  - Desarrollo de servicio web
    ?? x 9€/hora
  - Documentación
    ?? x 9€/hora
  - *Instancia que permita usar CUDA*
    En esta parte se aprovechará el servidor que ofrece la tutora del trabajo Maria Isabel García Arenas, con gran capacidad de cómputo y de manera gratuita. Esto ayudará de manera muy importante al trabajo, ya que reducirá de manera muy importante el presupuesto final del proyecto.
  - *Dominio para alojar una web publica del servicio*
    Al igual que en el punto anterior, se aprovechará el servicio de la tutora para lanzar el servicio web, reduciendo también el presupuesto final.

    Presupuesto total real: ? €

4.2. Tecnologías

En este apartado hablaremos sobre las tecnologías utilizadas en el trabajo. Se desglosarán después de listarlas:

En la parte del servidor:

- Python
- Django
- Apache
- C++
- CUDA

Y de cara al cliente:

- HTML5
- CSS3
- JavaScript
- AngularJS
- jQuery
- JSON
- HTML5UP

4.2.1. Servidor

- **Python**

  Versión usada: 2.7.6

  Python [32] es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible, además de simple y versátil.

  Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Figura 4.3: Logo de Python

- **Django**

  Versión usada: 1.10

  Django [10] es un framework de alto nivel web de Python. Fomenta el rápido desarrollo y un diseño limpio y pragmático. Se encarga de facilitar la creación de aplicaciones web complejas, ofreciendo las funcionalidades básicas y más útiles al desarrollador, para que pueda centrarse en la escritura de su aplicación.

Figura 4.4: Logo de Django

- Apache

  Versión usada: 2.4.12

  El servidor HTTP Apache [6] es un servidor web de código abierto, multiplataforma, extensible y modular. Además, al ser tan popular tiene un gran soporte de la comunidad.

  Implementa el protocolo HTTP y la noción de sitio virtual.

Figura 4.5: Logo de Apache

- **C++**

  Versión usada: 4.8.8

  C++ [8] es un lenguaje de programación que extiende a C añadiendo mecanismos que permiten la manipulación de objetos.

  Tiene las facilidades de programación genérica junto a los paradigmas de programación estructurada y programación orientada a objetos.

- **CUDA**

  Versión usada: 7.5

  CUDA [31] es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.

  Dispone de una gran comunidad, ya que es la arquitectura de cálculo paralelo más usada en la actualidad, en parte gracias a su potencia y la facilidad de uso para los desarrolladores.

  La plataforma de cálculo paralelo CUDA proporciona unas cuantas extensiones de C y C++ que permiten implementar el paralelismo en el procesamiento de tareas y datos con diferentes niveles de granularidad y con la flexibilidad de usar varios lenguajes, como son C, C++ y Fortran.

  Dispone de numerosas facilidades para los desarrolladores, como herramientas de desarrollo, documentación y su *toolkit*.

Figura 4.6: Logo de CUDA

4.2.2. Cliente

- **HTML5**

  HTML (HyperText Markup Language) [18] en su la quinta revisión, es el lenguaje básico de la World Wide Web. Es un estándar que sirve de referencia de elaboración de páginas web en sus diferente versiones. Define una estructura básica y un código para la definición de contenido de una página web, como texto, imágenes o vídeos. Usa un sistema para formatear el layout de nuestras páginas, así como hacer algunos ajustes a su aspecto. Los navegadores como Chrome, Firefox, Explorer o Safari puede saber como representar el contenido de una web, con las ubicaciones de todos los elementos.

Figura 4.7: Logo de HTML5

- **CSS3**

  CSS (Cascading Style Sheets) [9] u hoja de estilo en cascada, es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML2. El World Wide Web Consortium (W3C) es el encargado de formular la especificación de las hojas de estilo, al igual que del lenguaje HTML, que servirán de estándar para los agentes de usuario o navegadores.

  Aunque la información de estilo puede ser definida en el mismo documento HTML, la idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

- **JavaScript**

  JavaScript [22] es un lenguaje de programación interpretado, que se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

  Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz

Figura 4.8: Logo de CSS3

de usuario y páginas web dinámicas (aunque existe una forma de JavaScript del lado del servidor).

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Figura 4.9: Logo de JavaScript

- **AngularJS**

  Versión usada: 1.5.7

  AngularJS [5] es un framework de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Whatever (MVW), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

  La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página

a un modelo representado por las variables estándar de JavaScript, que además se pueden configurar manualmente, o recuperados de los recursos JSON estáticos o dinámico

Figura 4.10: Logo de AngularJS

- **jQuery**

  Versión usada: 1.11.3

  jQuery [24] es una biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

  jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript de manera muy reducida, por lo que con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio. Todo esto hace que sea la biblioteca JavaScript más usada.

Figura 4.11: Logo de jAuery

- **JSON**

  JSON (JavaScript Object Notation) [25] es un formato ligero de intercambio de datos. Gracias a su sencillez es fácil leerlo y escribirlo para humanos, mientras que para las máquinas es simple interpretarlo y generarlo.

  Es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de los principales lenguajes.

  Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

- **HTML5UP**

Figura 4.12: Logo de JSON

HTML5UP [19] Web con múltiples plantillas disponibles (usando HTML y JavaScript) y estilos CSS con los que crear la estructura básica de proyectos web. Son completamente personalizables y tienen un diseño web adaptable (*responsive*) de manera que la web que creemos se adaptará a cualquier pantalla, punto muy importante en la actualidad, debido a la multitud de dispositivos y tamaños de pantalla que existen.

Figura 4.13: Logo de HTML5UP

4.3. Herramientas

En este apartado se listarán y desglosarán las herramientas usadas para usar las tecnologías anteriores.

- **Nsight Eclipse Edition**

  Versión usada: 7.5

  Nsigh (NVIDIA Nsight Eclipse Edition) [27] es la gran plataforma de desarrollo para cálculo heterogéneo. Permite trabajar con potentes herramientas de depuración y análisis del rendimiento para optimizar el funcionamiento de la CPU y la GPU. Esta plataforma permite optimizar el rendimiento de manera intuitiva, identificar y analizar los cuellos de botella y observar el comportamiento de todas las actividades del sistema.

  Consta de varias plantillas y ejemplos, además de estar disponible varios sistemas operativos.

Figura 4.14: Logo de Nsight

- **TeXstudio**

  Versión usada: 2.11.0

  TeXstudio [37] es un entorno de escritura para la creación de documentos LaTeX de manera fácil, cómoda e intuitiva. Tiene numerosas características, como el resaltado de sintaxis, pre-visualizador integrado, verificación de referencias y varios asistentes.

  Tiene una gran documentación y comunidad, además de estar disponible para los sistemas operativos más importantes.

- **Git**

  Versión usada: 1.91

  Git [15] es un software de control de versiones, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

Figura 4.15: Logo de TeXstudio

Git se considera un sistema de control de versiones con funcionalidad plena y con multitud de características, como la gestión distribuida o la rápida gestión de ramas.

Figura 4.16: Logo de Git

- **Sublime Text**

  Versión usada: 2, versión de evaluación

  Sublime Text [36] es un editor de texto y de código fuente. Originalmente fue desarrollado como una extensión de Vim y con el tiempo fue creando una identidad propia.

  Poseee varias características para los desarrolladores, como un *minimapa* del código, multi-selección y multi-cursor, búsqueda dinámica y soporte a multitud de lenguajes.

- **Gimp**

  Versión usada: 2.8.10

  GIMP (GNU Image Manipulation Program) [14] es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías.

Figura 4.17: Logo de Sublime Text

Tiene multitud de herramientas para el retoque y edición de imágenes o un dibujo libre.

Esta disponible para la mayoría de los sistemas operativos y en varios lenguajes.

Figura 4.18: Logo de Gimp

- **GanttPRO**

  GanttPRO [13] permite la gestión de proyectos mediante diagramas de Gantt.

  Mediante su interfaz se pude planificar y gestionar proyectos con facilidad con diagramas de Gantt, mediante tareas, subtareas o duraciones dinámicas.

Figura 4.19: Logo de GanttPRO

- **Draw.io**

  Draw.io [11] es una herramienta para crear varios tipos de diagramas, entre ellos del tipo UML.

Figura 4.20: Logo de Draw.io

Capítulo 5

Descripción del sistema

En este capítulo se describe la fase de análisis del proyecto. Algunos requisitos o partes del capítulo fueron modificados o corregidos durante el trabajo. A continuación se muestran los definitivos.

5.1. Análisis inicial

Este proyecto fue propuesto desde un principio por Maria Isabel García Arenas, ya que en su ámbito de investigación requiere algoritmos evolutivos con muchos requerimientos de cómputo, por lo que al pararelizarlos se logran unos resultados más rápidos, y al estar disponible en un servicio web no es necesario un dispositivo específico.

Después de la propuesta del trabajo, y antes del desarrollo del mismo se hicieron diversas reuniones para establecer sus bases y alcance.

En la selección de tecnologías a usar se optó por la arquitectura de cómputo CUDA, ya que es actualmente la más potente, como se ve en el capítulo anterior, y se dispone de una tarjeta gráfica NVIDIA de gran potencia.

En la parte del servicio web se optó por el framework python Django, ya que se estudió con profundidad a lo largo del curso y es en la actualidad es la mejor opción para realizar un servicio web.

Dichas tecnologías son actuales, con una gran comunidad detrás, de manera que están en continuo desarrollo y mejora, además de estar bien asentadas, dando robustez al trabajo.

5.2. Objetivos

Este trabajo tiene como principal objetivo el desarrollo de unas clases que lancen algoritmos genéticos de forma paralela, y puedan ser usados por un servicio web.

Los principales objetivos, de manera reducida, que se quieren alcanzar con este trabajo son:

- **OBJ. 1** Gestión del servicio web
  - **OBJ. 1.1** Interfaz sencilla y adaptativa
  - **OBJ. 1.2** Recogida de parámetros
  - **OBJ. 1.3** Petición al servidor
  - **OBJ. 1.4** Recogida y muestra de la solución
- **OBJ. 2** Gestión del algoritmo genético
  - **OBJ. 2.1** Recogida de parámetros
  - **OBJ. 2.2** Ejecución del algoritmo
  - **OBJ. 2.3** Salida formateada

5.3. Resumen de implicados

Usuario

Descripción: Representa al usuario que quiere hacer uso del sistema.

Tipo: Usuario del sistema

Responsabilidad: Completar el formulario con los datos que requiera el algoritmo genético.

Administrador del sistema

Descripción: Representa a la persona encargada de mantener, actualizar y gestionar el sistema.

Tipo: Superusuario

Responsabilidad: Realizar todas las actividades de gestión y actualización del sistema.

5.4. Especificación de requisitos

5.4.1. Requisitos funcionales

En esta sección del capítulo se definirá y describiran las características de alto nivel (requisitos funcionales) del sistema que son necesarios para las necesidades del usuario.

RF 1 Gestión del servicio web

- RF 1.1 Despliegue de la web con sus elementos y formularios bien situados.
- RF 1.2 Recogida de parámetros mediante el formulario de la función que ha escogido el usuario.
- RF 1.3 Petición para ejecutar el algoritmo
- RF 1.4 Recogida y maquetado de la respuesta del algoritmo en la web

RF 2 Gestión del algoritmo genético

- RF 2.1 Recogida de parámetros
- RF 2.2.1 Ejecución del algoritmo genético evaluado con Ackley
- RF 2.2.1 Ejecución del algoritmo genético evaluado con Rastrigin
- RF 2.3 Salida formateada

5.4.2. Requisitos no funcionales

En esta sección se establecen los requisitos no funcionales.

Rendimiento

RNF 1: Será implementado con tecnologías que optimicen el rendimiento y la eficacia, tanto en el servicio web como en el algoritmo genético.

Disponibilidad

RNF 2: Se intentará que el servicio esté disponible las 24 horas del día, con una estructura segura y preparada ante los fallos.

Accesibilidad

RNF 3: Se logrará una gran accesibilidad al tratarse de un servicio web.

Usabilidad

RNF 4: La interfaz, con el orden y disposición de los elementos será lo más usable posible.

Interfaz

RNF 5: La interfaz será sencilla e intuitiva

RNF 6: Se adaptará a cualquier tamaño de pantalla, tendrá un diseño adaptativo.

Estabilidad

RNF 7: Se buscará la mayor estabilidad en el sistema, usando tecnologías robustas y estables.

Mantenimiento

RNF 8: Se ha desarrollado de manera ordenada y documentando todos las partes, para que el mantenimiento pueda ser realizado por el mismo desarrollador o por terceros de manera sencilla.

Soporte

RNF 9: Se le facilitará al usuario contacto con alguien responsable del sistema, para que en caso de necesitar soporto le sea fácil y rápido.

5.4.3. Requisitos de información

En esta sección se establecen los requisitos de información, que estarán fuertemente relacionados a los requisitos funcionales, ya que será la información mínima para llevarlos a cabo.

RI 1 Gestión del servicio web

- RI 1.1 Recogida de parámetros para el algoritmo genético
    - Tamaño de la población
    - Número de cromosomas

- Valor mínimo
- Valor máximo
- Probabilidad de cruce
- Probabilidad de mutación
- Número de generaciones
  - RI 1.1.1 Recogida de parámetros para la función Ackley
    - Valor A
    - Valor B
    - Valor C
  - RI 1.1.2 Recogida de parámetros para la función Rastrigin
    - Valor A

- RI 1.2. Recogida y maquetado de la salida del algoritmo genético
  - Rendimiento
  - Tiempo de cómputo
  - Tamaño del cómputo
  - Hebras usadas
  - Datos de entrada (RI 1.1)
  - Información del dispositivo que ha realizado el algoritmo
  - Mejores variables obtenidas
  - Mejor fitness

RI 2 Gestión del algoritmo genético

- RI 2.1 Recogida de parámetros
  - Dispositivo a usar
  - Función de optimización a usar
  - Tamaño de la población
  - Número de cromosomas
  - Valor mínimo
  - Valor máximo
  - Probabilidad de cruce
  - Probabilidad de mutación
  - Número de generaciones
    - RI 2.1.1 Recogida de parámetros para la función Ackley
      - Valor A

        - ◇ Valor B
        - ◇ Valor C
    - ○ RI 2.1.2 Recogida de parámetros para la función Rastrigin
        - ◇ Valor A

- RI 2.2. Salida formateada en JSON de la solución obtenida
    - Rendimiento
    - Tiempo de cómputo
    - Tamaño del cómputo
    - Hebras usadas
    - Datos de entrada (RI 2.1)
    - Información del dispositivo que ha realizado el algoritmo
    - Mejores variables obtenidas
    - Mejor fitness

5.4.4. Restricciones

Las restricciones del sistema serán las siguientes:

RI 1.1 Recogida de parámetros para el algoritmo genético:

- RSTR 1.1: El tamaño de la población será menor de 10.000.
- RSTR 1.2: Habrá un máximo de 1.000 cromosomas (1000 variables).
- RSTR 1.3: El valor máximo tendrá que ser más grande el el valor mínimo.
- RSTR 1.4: La probabilidad de cruce tendrá que ser entre 0 y 1.
- RSTR 1.5: La probabilidad de mutación tendrá que ser entre 0 y 1.

RI 2.1 Recogida de parámetros (del algoritmo genético)

- RSTR 2.1: Para escoger la función con la que optmizar introducimos 0 o 1.
- RSTR 2.2: El tamaño de la población será menor de 10.000.
- RSTR 2.3: Habrá un máximo de 1.000 cromosomas (1000 variables).
- RSTR 2.4: El valor máximo tendrá que ser más grande el el valor mínimo.
- RSTR 2.5: La probabilidad de cruce tendrá que ser entre 0 y 1.
- RSTR 2.6: La probabilidad de mutación tendrá que ser entre 0 y 1.

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] Algoritmo genético en Python. http://robologs.net/2015/09/01/como-programar-un-algoritmo-genetico-parte-ii-implementacion-en-python/

[2] Alojamiento web DonDominio. https://www.dondominio.com/

[3] Alojamiento web HostPapa. https://www.hostpapa.es/

[4] AMD OpenCL Accelerated Parallel Processing. http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/

[5] AngularJS. https://angularjs.org/

[6] Apache. https://httpd.apache.org/

[7] BrookGPU. http://graphics.stanford.edu/projects/brookgpu/

[8] C++. https://es.wikipedia.org/wiki/C%2B%2B

[9] CSS3. https://www.w3.org/TR/CSS/

[10] Django. https://www.djangoproject.com/

[11] Draw.io. https://www.draw.io/

[12] Framework rCUDA para ejecutar programas CUDA en un servidor externo. http://www.rcuda.net/index.php/what-s-rcuda.html

[13] GanttPRO.
https://app.ganttpro.com

[14] Gimp.
https://www.gimp.org/

[15] Git.
https://git-scm.com/

[16] 'Global Optimization Toolbox' para MatLab.
http://es.mathworks.com/products/global-optimization/

[17] gpuocelot: Framework dinámico de compilación que soporta CUDA.
https://code.google.com/archive/p/gpuocelot/

[18] HTML5.
https://www.w3.org/TR/html5/

[19] HTML5UP.
https://html5up.net/

[20] Impact: CUDA framework.
http://impact.crhc.illinois.edu/mcuda.aspx

[21] Instancia en Amazon Web Services que permita usar la GPU. https://aws.amazon.com/blogs/aws/new-ec2-instance-type-the-cluster-gpu-instance/

[22] JavaScript.
https://www.javascript.com/

[23] JGAP: Framework java para realizar algoritmos genéticos.
http://jgap.sourceforge.net/

[24] jQuery.
http://jquery.com/

[25] JSON.
http://www.json.org/

[26] Medias de sueldo para programadores Junior.
http://espana.jobtonic.es/salary/26526/16258.html?currency=EUR

[27] Nsight.
http://www.nvidia.es/object/nsight-es.html

[28] NVIDIA Developer.
https://developer.nvidia.com/, (Accessed on 01/07/2016)

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

[30] PeackStream comprado por Google.
https://www.crunchbase.com/organization/peakstream#/entity

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

[32] Python.
https://www.python.org/

[33] RapidMind es comprada por Intel.
https://software.intel.com/en-us/articles/intel-array-building-blocks

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

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

[36] Sublime Text.
https://www.sublimetext.com/

[37] TeXstudio.
http://www.texstudio.org/

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