uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática
2018-2019

Trabajo Fin de Grado

"Diseño e implementación de un selector dinámico de implementaciones basado en probabilidades"

Andrés Sánchez Cuadrado

Tutor

Javier Fernández Muñoz

RESUMEN

Debido a la progresiva adopción de plataformas heterogéneas como vía para incrementar el rendimiento de los sistemas de cómputo, ha surgido la necesidad de buscar soluciones que permitan el desarrollo de aplicaciones que empleen de forma eficiente todos los componentes que se encuentran en estos sistemas. A Causa de la variante naturaleza de estos elementos y sus diferencias en términos de rendimiento en función de qué operaciones se lleven a cabo, están siendo publicadas diferentes aproximaciones para aprovechar las capacidades de cada recurso y al mismo tiempo proporcionar a los desarrolladores herramientas que les facilite dicha tarea.

Con este trasfondo, el presente trabajó tratará de proporcionar una nueva solución que cubra estas necesidades gracias al desarrollo de un selector de implementaciones que fundamentará su algoritmo de selección en un planteamiento probabilístico desarrollado por el profesor de la Universidad Carlos III de Madrid Javier Fernández Muñoz.

De esta forma, dado un sistema heterogéneo donde se cuente con múltiples unidades de procesamiento de distinta naturaleza y un código fuente donde se especifique una implementación para cada unidad, la solución presentada proporcionará los mecanismos necesarios para la identificación de dichas implementaciones, los dispositivos sobre los que se han de ejecutar y finalmente seleccionará qué implementación es la más adecuada en términos de tiempo de ejecución para resolverla.

El desarrollo se llevará a cabo utilizando la metodología ágil SCRUM, y se realizará un análisis comparado con el planificador “versioning” del framework OmpSs para comprobar la eficacia del selector.

ABSTRACT

The solution to increase the performance in computer-based systems has been, in the recent years, the adoption of heterogeneous platforms, this decision produces the necessity of developing applications that make an efficient use of the different components in those systems. Since each element in this kind of platform produces a different performance depending on the operations assigned, new approximations are emerging to make a reasonable use of the resources and to give the developers tools to facilitate this task.

With this background, the present document tries to provide a new solution to cover these needs by means of a probabilistic selector, using a probabilistic model developed by professor Javier Fernández Muñoz from the Carlos III University.

Considering a heterogeneous system with multiple processing units and a source code with an implementation for each one, the proposed solution will identify the implementations, the corresponding devices and will select which implementation is the best in terms of execution time.

The development has been done following the SCRUM method and a comparative analysis with the "versioning" selector from the OmpSs framework shall be carried out to evaluate the selector.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

1. INTRODUCCIÓN

1.1. CONTEXTO DEL TRABAJO DE FIN DE GRADO

La finalidad del presente documento es exponer un sistema de selección de implementaciones basado en probabilidades. El selector incluye soporte para plataformas tanto homogéneas como heterogéneas.

Por ello es preciso definir en un primer momento qué son las plataformas heterogéneas, porqué es preciso invertir tiempo y esfuerzo en su análisis y cuáles son las razones que fundamentan el desarrollo de un selector de implementaciones para estas.

1.1.1. PLATAFORMAS HETEROGÉNEAS

El concepto de plataformas heterogéneas engloba a aquellos sistemas que cuentan con más de un tipo de procesador. Eso no implica que todos los sistemas con múltiples procesadores sean heterogéneos, ya que si todos los procesadores que utiliza son del mismo tipo se consideraría un sistema homogéneo. En cambio, todos los sistemas heterogéneos son multiprocesador, pero estos procesadores que lo definen son de diferente naturaleza.

A la hora de distinguir diferentes tipos de procesadores, se suele a recurrir a su funcionalidad principal que conlleva una arquitectura diferente. Por ejemplo, los procesadores vectoriales se desarrollan para realizar cálculos repetitivos sobre grandes cantidades de datos reduciendo el proceso de obtención de instrucciones, los procesadores de señales digitales se enfocan en el procesamiento de señales continuas, etc. [1] De esta forma, cada procesador cuenta con un flujo de trabajo diferente.

Cada tipo de procesador cuenta con una serie de virtudes y desventajas que hacen que sean los sistemas adecuados para una serie de funcionalidades concretas. Con esta premisa, los sistemas heterogéneos tratan de integrar en un mismo sistema diferentes procesadores de forma que se emplee cada uno cuando sus características le hagan el más adecuado para la tarea a realizar.

Es en este punto cuando un selector de implementaciones cobra sentido, debe contarse con una solución que permita identificar qué procesador es el más adecuado para cada tarea de forma que se aproveche la arquitectura heterogénea.

1.1.2. FRAMEWORKS

A la hora de desarrollar el sistema de selección de implementaciones, se hará uso de ciertos frameworks de programación paralela, por lo que es importante definir qué se entiende por framework y sus elementos más significativos.

El concepto de framework se traduce al castellano, en el contexto informático, como un entorno o marco de trabajo y consiste en un conjunto de herramientas orientadas a la realización de una tarea o un conjunto específico de estas. Los factores clave que definen y diferencian a los frameworks de otros elementos como por ejemplo librerías son [2] [3]:

- Reutilizable: tanto el diseño como la implementación debe poder ser reutilizado por diferentes usuarios.
- Extensible: debe ser posible la introducción, por parte del usuario, de nuevas funcionalidades.
- Flujo de trabajo predeterminado: uno de los elementos que diferencian al framework de una librería es que el flujo de trabajo no es controlado por el usuario, si no que el propio framework establece cómo debe ser utilizado para su correcto funcionamiento. El usuario define funciones que serán llamadas por el framework, lo cual es exactamente lo contrario al uso de librerías, donde es el usuario quien llama a las funcionalidades que esta proporciona.
- Núcleo no modificable: El usuario debe poder implementar funcionalidades que serán utilizadas por el framework, sin embargo, este no debería modificar el funcionamiento interno del framework, solo ampliar o definir parte de este según lo establezca el propio framework.

Dentro de los frameworks de programación nos encontramos con aquellos que permite la definición de diferentes implementaciones para una misma función. Gracias a esta característica es posible contar, dentro de un único programa, con múltiples variaciones de código que se adapten a necesidades diferentes, entre ellas se puede destacar el tamaño de los parámetros que recibe la función, la eficiencia en términos de energía, eficiencia en tiempo de ejecución, contar con diferentes versiones en función de los dispositivos con los que cuenta la máquina, etc.

Para ello, el framework debe poner a disposición del usuario las herramientas para definir las diferentes implementaciones y a su vez proporcionar un criterio a la hora de elegir qué implementación se va a utilizar en la ejecución del programa. En este punto entran en juego los selectores de implementaciones.

1.1.3. SELECTOR DE IMPLEMENTACIONES

Un selector de implementaciones consiste en un programa o fragmento de este que recibe como parámetros de entrada diversas versiones de una función y contiene una lógica que permite determinar qué implementación será elegida para su ejecución.

Dependiendo de la implementación del selector, se puede contar con uno o varios algoritmos según los cuales se decide la implementación a ejecutar y estos se fundamentan en diferentes objetivos como maximizar la eficiencia en términos de tiempo de ejecución, minimizar el consumo energético, evitar tener dispositivos sin carga de trabajo, etc.

1.1.4. TAREAS

Respecto al selector de implementaciones, es importante definir el concepto de tarea o "task", ya que es la unidad sobre la que se realiza la selección por parte del planificador.

Este término presenta diferentes significados dependiendo del contexto en el que se utilice. De cara al procesador, una tarea sería la unidad básica que el procesador es capaz de controlar. Desde un punto de vista más general, tarea podría ser sinónimo de proceso o hilo (thread), donde un proceso es la instancia de un programa en ejecución y por hilo se entendería la unidad básica que un planificador puede manejar y forma parte de un proceso. No obstante, este apartado no trata de definir qué es un proceso o un hilo, sino qué es una tarea desde el punto de vista de un planificador.

En lo que a este trabajo se refiere utilizaremos la definición proporcionada en la documentación de OpenMP. Una tarea es la unidad mínima de ejecución que puede ser gestionada de forma independiente por un planificador. Más concretamente, las tareas son bloques de código que son identificados como tales por medio de algún mecanismo, como por ejemplo una anotación en el código, y el compilador es capaz de detectarlas y ejecutarlas en paralelo [4].

1.1.5. ALGORITMOS DE PLANIFICACIÓN

Una vez definidas las tareas el siguiente paso es determinar en qué procesador se va a ejecutar cada una. Para ello entran en juego los algoritmos de planificación.

Un algoritmo, de forma genérica, consiste en una serie de pasos para resolver un problema. El problema, en este caso, es la distribución de tareas en dispositivos de forma que se obtenga una mayor eficiencia en algún aspecto concreto. Uno de los objetivos clave sobre los que enfocar la eficiencia puede ser el tiempo de ejecución de un programa, sin embargo, es posible establecer otros, como puede ser la utilización de recursos energéticos.

En este contexto, el presente documento pretende aportar:

1) Un selector de implementaciones que basándose en una formula probabilística escoja entre una serie de algoritmos, cuál es el más adecuado empleando información de ejecuciones previas.
2) Exponer el entorno de ejecución del planificador y su integración con el framework OmpSs.
3) Se realiza una evaluación de la eficiencia del selector empleando la multiplicación de matrices como caso de uso.

1.2. OBJETIVOS DEL PROYECTO

El objetivo principal del presente trabajo consiste en el desarrollo de una herramienta que dé respuesta al desafío que plantean los sistemas heterogéneos, por lo que deberá tener en cuenta los siguientes aspectos:

1. Debe ser capaz de identificar diferentes implementaciones de una misma función
2. Debe ser capaz de seleccionar de forma dinámica la implementación más eficiente en función del problema a resolver
3. Debe ser capaz de almacenar la información obtenida en ejecuciones anteriores para llevar a cabo la selección de implementación
4. Debe proporcionar al usuario una interfaz intuitiva para su uso

El primer objetivo se fundamenta directamente en la naturaleza de los sistemas heterogéneos, donde conviven diferentes procesadores cuya idoneidad dependerá del problema en cuestión, por lo que es necesario en estas plataformas la integración de diferentes implementaciones de una misma función para cada tipo de procesador.

El segundo objetivo pretende que la existencia de diferentes procesadores, de diferente naturaleza, en un mismo sistema proporcione una ventaja en términos de tiempo de ejecución a la hora de resolver un problema, ya que, en función de circunstancias diferentes, un procesador u otro puede encontrar la solución de forma más eficiente.

El tercer objetivo trata de reducir el tiempo necesario para la selección de la implementación más eficiente empleando información que se ha obtenido previamente, reduciendo así el número de ejecuciones necesarias para seleccionar la implementación más eficiente.

El cuarto objetivo se basa en un requisito fundamental a la hora de ofrecer una herramienta que busca ser utilizada de forma generalizada por los usuarios a la que va dirigida. En este caso, a desarrolladores que trabajen en plataformas heterogéneas. Es por ello por lo que proporcionar una forma de emplear la herramienta que resulte lo más compresible posible favorecerá la incorporación de la misma a los proyectos en los que pueda ser de utilidad.

Para satisfacer los objetivos anteriores, se plantea el desarrollo de un seleccionador de implementaciones basado en probabilidades. El modelo probabilístico que se empleará fue desarrollado por el profesor Javier Fernandez Muñoz.

1.3. ESTRUCTURA DEL DOCUMENTO

Estado de la cuestión o contexto tecnológico

En primer lugar, se realizará un ejercicio de investigación para determinar cuál es el contexto tecnológico sobre el que se fundamente este trabajo. Se analizarán qué herramientas existen para solucionar los problemas que se buscan atajar con el seleccionador a desarrollar y se indicarán qué ventajas e inconvenientes presentan.

Metodologías ágiles

El siguiente punto expondrá en qué consiste la metodología SCRUM y se desarrollará los elementos que la componen, así como los beneficios que proporciona.

Análisis del sistema

Esta sección recoge cómo se ha integrado la metodología SCRUM en este proyecto y cuenta con los recursos que propone para el planteamiento, seguimiento y realización del proyecto.

Diseño del sistema

En este apartado se desarrolla el núcleo del trabajo, se definen los elementos que componen al selector de planificaciones, las modificaciones que se han hecho en el framework de OmpSs para su incorporación y se explica el algoritmo de selección basado en probabilidades.

Evaluación de resultados

Con el objetivo de comprobar la eficacia del planificador, se lleva a cabo un análisis de este, realizando un ejercicio de multiplicación de matrices de diferentes tamaños para comprobar su efectividad y a continuación, se realiza una comparativa con el planificador "versioning" de OmpSs para comprobar si la alternativa presentada ofrece alguna mejora.

Planificación, presupuesto y entorno socioeconómico

Se realiza un estudio de los costes asociados al proyecto y la planificación que se va a seguir en su desarrollo. A su vez, se discuten las implicaciones del resultado del proyecto en lo que respecta al ámbito económico, social y medioambiental.

Marco regulador

Se recogen aquellas regulaciones que afectan al presente trabajo, en qué medida afectan y cómo se han abordado.

Conclusiones

Para finalizar el trabajo se ofrecen una serie de conclusiones sobre el proyecto, seguidas de las impresiones personales que se han percibido durante la realización del mismo y finalmente se apuntan algunos aspectos que podrían mejorarse o desarrollar en trabajos futuros.

2. ESTADO DEL ARTE

2.1. INTRODUCCIÓN

En esta sección se va a exponer cuál es la situación actual de las arquitecturas heterogéneas, cuál es su objetivo principal, sus ventajas y algunas de las soluciones que existen actualmente para abordarlas.

2.2. CONTEXTO TECNOLÓGICO

En la actualidad nos encontramos en un marco tecnológico donde se ha alcanzado un estancamiento en lo que a procesadores de un solo núcleo se refiere. La frecuencia a la que estos trabajan se ha visto limitada por diversos factores, entre ellos, la temperatura a la que se ven sometidos al trabajar a ciertas frecuencias y la energía que es requerida en estas condiciones, este factor es conocido como "power wall" o muro de energía [5].

Este fenómeno se puede apreciar en la siguiente imagen, donde se puede observar como la frecuencia de los procesadores ha visto reducido su incremento desde 2004 aproximadamente.

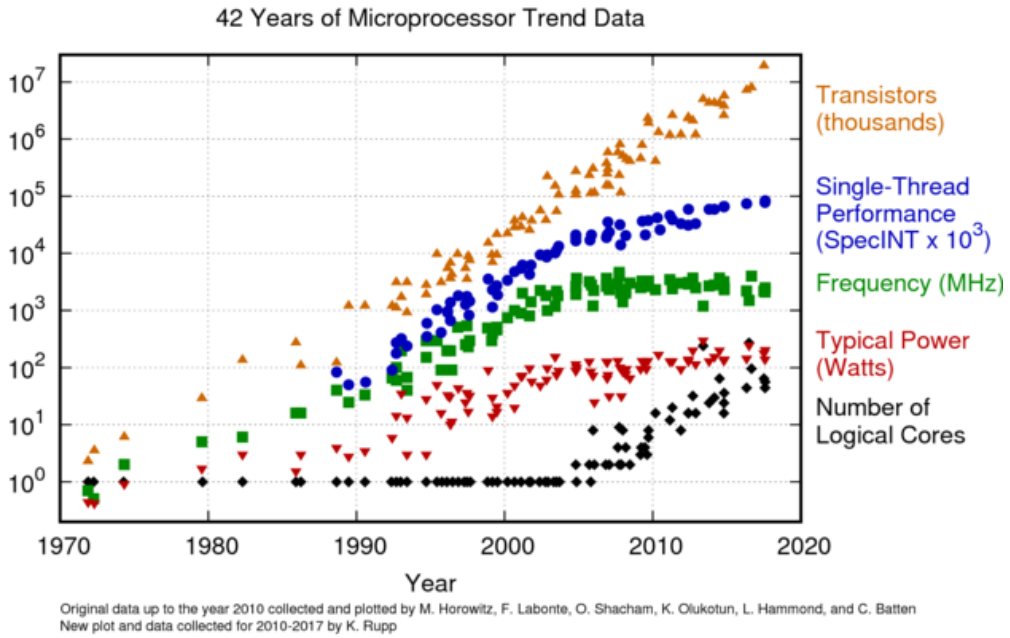


Figura 1: 42 Years of Microprocessor Trend Data. [33]

Como respuesta, los fabricantes han ido aumentando progresivamente el número de núcleos de procesamiento, de esta forma, se continúa experimentando un incremento en el rendimiento de los mismos. Al mismo tiempo, no solo ha crecido el número de núcleos por unidad de procesamiento, si no que los dispositivos cuentan con una variedad de diferentes unidades destinadas a realizar tareas diferentes, encontrándonos así ante lo que se han denominado plataformas heterogéneas.

En los últimos años se ha popularizado la utilización de arquitecturas heterogéneas, tanto para la resolución de problemas de ámbito científico e industrial, como en aplicaciones de uso diario. En la actualidad, gran parte los ordenadores de uso doméstico cuentan con GPUs para realizar tareas que requieren frecuentes operaciones de coma flotante.

De esta forma, se desarrollan diferentes implementaciones de un mismo algoritmo en función de la plataforma que lo va a utilizar. Alguno de estos ejemplos se puede encontrar en librerías como BLAS [6] cuyo objetivo es proporcionar un set de rutinas de bajo nivel para realizar operaciones comunes de algebra lineal. De esta librería se pueden encontrar distintas versiones, como clBLAS [7], optimizada para OpenCL, o cuBLAS [8] optimizada para CUDA.

En este escenario se puede entrever la necesidad de contar con herramientas que faciliten la elección entre distintas implementaciones para una misma funcionalidad. Estas soluciones se pueden categorizar como:

Estáticas

Estos planificadores realizan un análisis en tiempo de compilación basado en ejecuciones anteriores para identificar cuál es la implementación y el dispositivo más adecuado para el problema a resolver.

En ejemplo de esta categoría se puede encontrar en el trabajo de Jun et al. [9] que plantean un análisis del código fuente para determinar cuál es la implementación más adecuada basándose en optimización de algebra linear.

En tiempo de ejecución

Estas soluciones emplean planificadores que actúan mientras el programa se está ejecutando para determinar cuál es el algoritmo y el dispositivo idóneo para el problema que se está abordando.

Esta alternativa es la que sigue el framework de OmpSs [10] con el planificador versioning [11], el cual escoge la implementación más adecuada entre una serie de opciones que son marcadas previamente por el desarrollador como implementaciones alternativas.

En esta corriente, otra alternativa la ofrece el framework SkePU [12] el cuál emplea algoritmos de aprendizaje automático para seleccionar una implementación u otra de una función en base a su tiempo de ejecución.

2.2.1. OPENMP

OpenMP [4] es una API (Interfaz de programación de aplicaciones) enfocada a la paralelización de programas empleando para ello una serie de directivas que reciben el nombre de *pragmas*. Gracias a estas directivas el usuario define que partes del código se paralelizan y es el responsable de utilizar OpenMP de forma que se eviten los problemas asociados a la programación en paralelo como conflictos de datos, condiciones de carrera (race condition) o bloqueos mutuos (deadlock).

OpenMP se puede utilizar con los lenguajes de programación C, C++ y Fortran, y puede emplearse en el desarrollo de aplicaciones para diversos sistemas operativos como Windows, Linux, macOS entre otros.

La última versión estable de OpenMP en el momento de la redacción es la versión 5.0 publicada en noviembre de 2018.

En relación al proyecto presentado en este documento, el aspecto fundamental de OpenMP es su conjunto de directivas, que van a ser utilizadas y ampliadas por el framework OmpSs y nos servirá de base para crear una nueva directiva que permita al planificador identificar las diferentes implementaciones. Las directivas de OpenMP siguen el siguiente formato en C y C++:

```
#pragma omp directive-name [clause[[,] clause]...]new-line
```

Figura 2: Pragma OpenM. [4] página 38

Como punto a destacar, las directivas siguen el estándar de C y C++ y se distingue entre el uso de mayúsculas y minúsculas.

Con esto presente, se buscará desarrollar la directiva que permita identificar las diversas implementaciones para el planificador basado en probabilidades siguiendo este formato y teniendo en cuenta uno de los objetivos que se quiere alcanzar con esta herramienta, la facilidad de uso por parte del usuario. Por tanto, tanto el número de cláusulas y su nombre deberá ser auto explicativo.

2.3. OMPSS FRAMEWORK

El framework OmpSs, desarrollado por el Barcelona Supercomputing Center, que trata de aunar las características del modelo de programación StarSs, también desarrollado por esta organización. El objetivo de este framework es ampliar las funcionalidades del API OpenMP. Dentro de las características de OpenMP, OmpSs va a hacer uso de las anotaciones denominadas "pragma" que permiten la implementación de diversas operaciones de una forma sencilla y legible. Un ejemplo de ello sería la ejecución de un fragmento de código en multihilo añadiendo la siguiente anotación:

```
#include <stdio.h>
#include <omp.h>
int main() {
    int x = 2;
#pragma omp parallel num_threads(2) shared(x)
    {
        if (omp_get_thread_num() == 0) {
           x = 5;
        } else {
           /* Print 1: the following read of x has a race */
           printf("1: Thread# %d: x = %d\n", omp_get_thread_num(), x);
        }
#pragma omp barrier
        […]
    }
}
```

Figura 3: OpenMP Example. [34]

De esta forma el código que se encuentra en el bloque de la anotación "pragma" se ejecutara en un máximo de dos hilos. Como se puede apreciar, se proporciona una forma clara e intuitiva de introducir funcionalidades al código que de otra forma supondría una estructura diferente y posiblemente más compleja al menos en términos de legibilidad y extensión.

Siguiendo esta corriente, el framework OmpSs introduce un set adicional de pragmas que permiten añadir funcionalidades al código siguiendo esta filosofía. Un ejemplo interesante de ello lo proporcionan los pragmas que permiten definir la dependencia de datos entre diferentes tareas. A continuación, se puede observar una imagen que contiene una implementación de esta definición de dependencias:

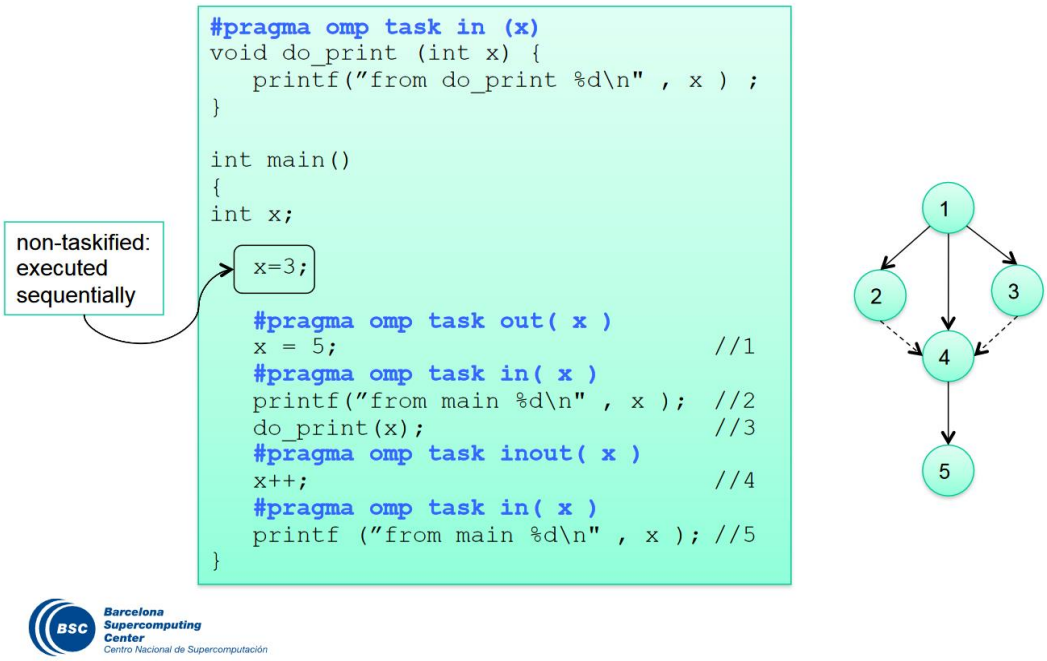


Figura 4: OmpSs Example. [35]

En el ejemplo anterior, se tiene como datos una única variable entera "x", pero podría tenerse un conjunto de variables de diferente tipo. Se puede apreciar que hay tres tipos de pragmas involucrados, el pragma "in" indica que se va a recibir como argumento de entrada la variable "x". El pragma "out" expresa que la variable "x" es un argumento de salida. Finalmente, el pragma "inout" indica que "x" es tanto un argumento de entrada como de salida.

No obstante, para el tema que ocupa el presente trabajo, el pragma más relevante es el que permite definir diferentes implementaciones de una misma función en base a un dispositivo. A continuación, se muestra un breve ejemplo de este pragma:

```
#pragma omp task out(v0) label(myFunction)
void myFunction(){/*...*/}

#pragma omp target device (Device1) implements(myFunction)
#pragma omp task out(v0) label(myFunctionAltVer)
void myFunctionAlternativeImplementation(){/*...*/}
```

Figura 5: OmpSs Pragma Example

En este ejemplo, se define una función llamada “myFunction” que tiene un argumento de salida “v0” y se indica que su nombre identificativo va a ser “myFunction” a través de la palabra “label” en el pragma.

Seguidamente, se define una función llamada “myFunctionAlternativeImplementation” que va a ser una implementación alternativa de la función definida previamente. Se indica a su vez que va a tener un argumento de salida “v0” y su nombre identificativo es en este caso “myFunctionAltVer”. Es sobre esta función donde se aplica el pragma que va a indicar que es una implementación alternativa de la función original con la línea “`#pragma omp target device (Device1) implements(myFunction)`” donde se expresa que el dispositivo que lo va a ejecutar es “Device1” y que implementa a la función “myFunction”.

Gracias a estos pragmas, podemos indicar una serie de implementaciones alternativas que afectan a distintos dispositivos, definiendo así un programa que saque el máximo partido a un sistema heterogéneo.

2.3.1. MERCURIUM

Mercurium [13] es un compilador source-to-source desarrollado por Barcelona Supercomputing Center. Los compiladores source-to-source son aquellos que parten de un código fuente escrito en un lenguaje de programación y producen un código fuente diferente, que puede estar definido en otro lenguaje de programación o en el mismo tras hacer ciertas modificaciones en este. Por norma general, los compiladores tradicionales suelen traducir un código escrito en un lenguaje de alto nivel a otro de bajo nivel, mientras que los compiladores source-to-source producen un código que se encuentra en un nivel de abstracción similar al código fuente adicional. A su vez, estos compiladores source-to-source se pueden emplear para actualizar código escrito en una API que se ha quedado obsoleta y producir su equivalente en la nueva versión de la interfaz.

Mercurium por su parte, admite los lenguajes C, C++ y Fortran. Su principal utilidad consiste en implementar OpenMp sobre el entorno de ejecución Nanos, lo cual implica traducir las directivas de OmpSs en llamadas a Nanos. Sin embargo, su utilidad es mayor gracias a que ha sido diseñado de forma modular, por lo que es posible añadir funcionalidades adicionales al compilador aumentando su versatilidad.

Los módulos o plugins de los que se compone Mercurium están desarrollados en C++ y se cargan de forma dinámica en base a la configuración que se defina.

Para el presente trabajo se incluirá un nuevo pragma para definir el tamaño del problema a resolver por lo que será preciso modificar el compilador Mercurium para que reconozca esta nueva funcionalidad.

2.3.2. NANOS++

Nanos++ [14] es un entorno de ejecución desarrollado por Barcelona Supercomputing Center. Nanos se utiliza principalmente con el compilador Mercurium para implementar OmpSs y una serie de servicios que permiten usar un entorno de programación paralela basado en tareas cuyo mecanismo de sincronización está basado en la dependencia de datos. Las tareas se implementan como hilos a nivel de usuario y se utilizan mecanismos basados en caché para mantener coherencia de memoria entre diferentes espacios de memoria, como pueden ser distintas GPUs.

Sin embargo, una de las virtudes principales de este entorno de ejecución consiste en la utilización de una variedad de políticas de planificación de tareas que pueden ser cargadas de forma dinámica en tiempo de ejecución, permitiendo la selección tanto del orden de ejecución de las tareas como del procesador en el que se van a ejecutar. Estas políticas de planificación se definen en módulos independientes, por lo que es posible desarrollar nuevas instancias para su utilización con Nanos.

Este documento presentará una nueva política de planificación que será utilizada en el entorno de ejecución Nanos++, por lo que será preciso su modificación.

2.4. CUDA

CUDA son las siglas de “Compute Unified Device Architecture” [15] o Arquitectura Unificada de Dispositivos de Cómputo, es una plataforma de computo paralelo y un modelo de programación desarrollado por la compañía NVIDIA que permite el desarrollo en plataformas heterogéneas al ofrecer soporte para el desarrollo de CPU y GPU. Es importante destacar que las GPU compatibles con esta plataforma son únicamente aquellas desarrolladas por la propia compañía.

CUDA, en el momento de la redacción de este documento, ofrece soporte para los leguajes de programación C, C++, C#, Fortran, Java y Python entre otros. [16]

La base de la programación en CUDA es relegar en la CPU todo el trabajo secuencial del código mientras que la parte paralelizable sea trasladada a los núcleos de la o las GPU, ya que se cuenta con soporte para sistemas con múltiples GPU. De esta forma, se da soporte a plataformas heterogéneas con múltiples unidades de procesamiento de diferente naturaleza.

2.5. OPEN-CL

Open-CL se refiere a “Open Computing Languaje” o Lenguaje de Computación Abierto y es un framework que permite desarrollar programas dirigidos a plataformas heterogéneas. Con este propósito, incluye una interfaz de programación de aplicaciones (API), un lenguaje de programación, una colección de librerías y un sistema de runtime para desarrollo. [17]

A diferencia de CUDA, Open-CL no se limita a las GPU de una compañía en particular, si no que busca ser compatible con una pluralidad de dispositivos. Otra diferencia respecto a CUDA es que es código abierto y libre, por lo que es compatible con cualquier dispositivo que se acomode al estándar que ofrece.

De esta forma, Open-CL permite el desarrollo en plataformas heterogéneas, al poder crear programas que pueden ejecutarse en máquinas que cuenten con múltiples unidades de procesamiento que pueden o no ser del mismo tipo.

2.6. SOLUCIONES ESTÁTICAS

2.6.1. CODE TRANSFORMATION FRAMEWORK

La propuesta de la universidad tecnológica de Nanyang [9], situada en Singapur, nos muestra un ejemplo de soluciones estáticas en cuanto a la selección de implementaciones.

Su punto de partida se basa en las aproximaciones fundamentadas en directivas, como OpenMP y OpenACC. Estos modelos permiten al desarrollador paralelizar regiones de código una vez que el propio desarrollador las ha identificado por su cuenta. Este utilizará las directivas para anotar dichas secciones y el compilador generará posteriormente código que permitirá ejecutar el contenido de forma simultánea gracias a diversos núcleos de ejecución. OpenMP lo hará enfocándolo a código para CPU mientras que OpenACC lo generará para GPU o CPU.

El problema radica en que el código generado no está optimizado para la arquitectura concreta que se está utilizado, por ejemplo, si se utilizan estas directivas para paralelizar un bucle y se está utilizando una GPU Nvidia, el código resultante no será óptimo para esta GPU, mientras que si se utiliza la librería CUBLAS, sí se haría uso de las características propias de arquitectura en la que se va a ejecutar este código, de forma que el programa resultante será más eficiente.

Con esta premisa, los investigadores de la universidad de Nanyang, proponen un framework para la transformación de código "source-to-source" capaz de analizar el código fuente original y generar uno nuevo que haga uso de las optimizaciones que los fabricantes de hardware ponen a disposición de los desarrolladores.

La solución cuenta con dos motores, el motor de extracción y el motor de mapeado, que van a permitir reconocer los patrones de las operaciones algebraicas que se encuentren en el código fuente, y posteriormente sustituirlas por llamadas específicas a las librerías correspondientes al hardware que se esté utilizando.

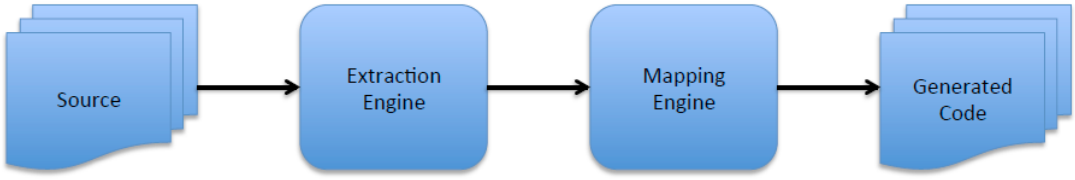


Figura 6: Elementos framework SkePU. [9] Página 5.

El factor clave de este framework es que hace un análisis en tiempo de compilación para detectar estos patrones que sustituirá por fragmentos de código optimizado. Esto implica que el análisis sucesivo del mismo programa generará siempre el mismo código fuente resultante, lo cual puede suponer que el código termine siendo subóptimo si el entorno o el problema que se está resolviendo cambia.

Por esta razón, la imposibilidad de adaptarse y de utilizar la información obtenida en diversas ejecuciones del programa, nos llevan a buscar otras soluciones que entran dentro del marco de propuestas en tiempo de ejecución.

2.6.2. CID: COMPILE-TIME IMPLEMENTATION DECIDER FOR HETEROGENEOUS PLATFORMS BASED ON C++ ATTRIBUTES

Una de las soluciones estáticas más interesantes viene de parte de la Universidad Carlos III de Madrid, por parte de los investigadores Luis Miguel Sánchez, David del Río Astorga, Manuel F. Dolz y Javier Fernández Muñoz [18].

En este caso, se propone una herramienta para decidir qué implementación utilizar en tiempo de compilación y en qué dispositivo se debe ejecutar. A su vez, se reutiliza la información obtenida en ejecuciones anteriores para la mejorar la decisión en futuras ejecuciones.

Se utilizan dos componentes para dar vida a esta herramienta, por una parte, se utilizan los atributos de C++11 y por otra, se emplea un lenguaje para la descripción de funcionalidades en plataformas heterogéneas llamado "Hardware Parallel Platform Description Language" (HPP-DL).

Los atributos de C++11 son unas etiquetas predefinidas que proporcionan información adicional al compilador. Con esta información, el compilador puede generar mensajes de información o aplicar cierta lógica al compilar el atributo. Destacar que es preciso que el compilador debe reconocer estos atributos ya que serán ignorados en cualquier otro caso. Otro punto por destacar de los atributos es que pueden incorporarse a varios elementos del código como tipos, variables y bloques entre otros. Sin embargo, los atributos no pueden ser definidos por el usuario en tiempo de ejecución e incorporar nuevos atributos conlleva la modificación del compilador de C++. Otra limitación de los atributos de C++ es que la información de los atributos no puede obtenerse desde la aplicación de usuario ya que el identificador de tipos en tiempo de ejecución de C++ (Run-Time Type Identification, RTTI) no preserva esta información.

Debido a estas limitaciones los autores hacen uso de los atributos desarrollados en el proyecto REPARA [19] para utilizar el compilador LLVM C++, de forma que se puedan utilizar para que el usuario pueda definir los requisitos necesarios a la hora de ejecutar una función y para especificar las propiedades de cada posible implementación.

En cuanto al lenguaje de descripción para hardware HPP-DL, mencionar que surge como parte del proyecto REPARA. Para cumplir su cometido, define tres clases elementos, por un lado "Components", son los componentes de hardware que tiene la plataforma e incluye las CPU, GPU, memoria principal y núcleos, por otra parte define los "Links" que van a establecer las relaciones entre dos componentes y dan información sobre la transmisión de datos, por último se definen los "Resources" que representan las interfaces por las que se accede a los dispositivos, en este elemento entrarían los puertos de entrada/salida.

Siguiendo este lenguaje, es posible generar archivos en formato JSON que permiten de una forma entendible para un ser humano definir los componentes de la plataforma. Por ejemplo, la siguiente imagen muestra un ejemplo de este tipo de archivos.

```
{ "class" : "hpp",
  "components" : [ {
        "class" : "platform",
        "id" : "platform_0",
        "description" : "System XXXX", ...
        }, {
        "class" : "processor",
        "id" : "platform_0.processor_0",
        "description" : "Intel XXXX", ...

        }, {
        "class" : "core",
        "id" : "platform_0.processor_0_core_0",
        "description" : "XXX", ...
        }, {
        "class" : "cache",
        "id" : "platform_0.processor_0.cache_0",
        "description" : "L1", ...
        "size" : "32 KiB", ...

   }, ...
 ],
 "links" :[ {
         "class" : "link",
         "description" : "link_processor_0_cache_0",
         "src_component" : "platform_0._processor_0.core_0"
         "dst_component" : "platform_0._processor_0.cache_0"
         },
  ]
}
```

Figura 7: Ejemplo HPP-DL

Con estas dos herramientas la solución presenta el siguiente flujo de trabajo:

1. Se obtienen las especificaciones de hardware en los documentos HPP-DL
2. El administrador del sistema heterogéneo define una serie de restricciones para cada interfaz e implementación, de forma que se especifica qué implementación está asociada a qué interfaz y a qué dispositivo. Esto se realiza en las cabeceras del código C++ y hay dos atributos para ello, uno define que fragmento del código es una implementación para una interfaz (rpr::implements) y el otro atributo indica a qué dispositivo corresponde la implementación (rpr::device).

```
namespace namespaceImplementation1 {
  [[rpr::implements("functionCall"), rpr::device(CPU)]]
  void functionCallImpl1(...);
}
namespace namespaceImplementation2 {
  [[rpr::implements("functionCall"), rpr::device(CPU)]]
  void functionCallImpl2(...);
}
```

Figura 8: Ejemplo atributos "implements" y "device"

3. El desarrollador deberá definir las llamadas a funciones de forma que CID pueda analizarlas. Para ello se utilizan principalmente dos atributos, uno define que la función es una interfaz y debe ser analizada y remplazada (rpr::cid) mientras que el segundo atributo indica el dispositivo en el que debe ejecutarse (rpr::target(device)). A su vez, se cuenta con tres atributos adicionales para indicar el tamaño del problema, ya sea el valor exacto (rpr::size) o el mínimo y máximo tamaño (rpr::minsize, rpr::maxsize).

```
#include <header.h>
int main(){
      [[rpr::cid, rpr::minsize(2.5), rpr::maxsize(5)]]
      functionCall(...);
 return 0;
```

Figura 9: Ejemplo atributos "cid", "minsize" y "maxsize"

4. Se aplica el algoritmo de selección. La herramienta CID analiza las funciones definidas con el atributo “rpr::cid” y comienza el proceso de selección en el que se reemplazan las interfaces por las implementaciones. Para determinar qué implementación es la más adecuada se utiliza el tamaño del problema y un histórico en el que se almacena los tiempos de ejecución pasados, dicho histórico se almacena en un documento JSON denominado PERF.json.

Los puntos fuertes de esta herramienta estarían en la sencilla y legible forma de expresar los componentes del sistema heterogéneo y sus relaciones a la vez que usar unos atributos sencillos para permitir la selección de diversas implementaciones en función del tamaño del problema y reutilizando la información de ejecuciones anteriores.

Como elementos a mejorar que se pretende cubrir en el presente trabajo, se encuentra la necesidad de modificar el compilador para incluir los nuevos atributos y la imposibilidad de cambiar la selección en tiempo de ejecución.

2.7. SOLUCIONES EN TIEMPO DE EJECUCIÓN

2.7.1. SKEPU: AUTOTUNABLE MULTI-BACKEND SKELETON PROGRAMMING FRAMEWORK FOR MULTICORE CPU AND MULTI-GPU SYSTEM

SkePU es un framework de programación desarrollado en C++ enfocado a sistemas que incorporan Unidad Graficas de Procesamiento (GPU) al mismo tiempo que Unidades Centrales de Procesamiento (CPU). Surge a raíz de un proyecto de investigación llevado a cabo en la universidad de Linköping, Suecia.

Su principal función es facilitar el desarrollo de aplicaciones con la ayuda de esqueletos. Estos esqueletos se definen como un componente predefinido que se basa en un patrón de computación y dependencia de datos y puede ser modificado mediante parámetros por el usuario.

Entre estos patrones se encuentra por ejemplo el patrón "map" el cual que recibe una serie de elementos a los que se somete una función que proporciona el usuario. Los desarrolladores proponen el siguiente ejemplo en el que se realiza una suma entre los elementos de dos vectores con el mismo índice, generando un vector del mismo tamaño con el resultado de la suma en el índice correspondiente.

```
float sum(float a, float b){
      return a + b;
}
Vector<float> vector_sum(Vector<float> &v1, Vector<float> &v2){
      auto vsum = Map<2>(sum);
      Vector<float> result(v1.size());
      return vsum(result, v1, v2);
}
```

Figura 10: Ejemplo patrón "map" SkePU [36]

El valor añadido que aporta SkePU a esta serie de patrones es proporcionar una variedad de implementaciones de los mismos para diferentes plataformas. Para el desarrollo secuencial, da soporte a C, OpenMP, OpenCL y CUDA. Para el desarrollo en GPU tiene soporte para OpenCL y CUDA.

De esta forma, permite la fácil adaptación del código en función de los elementos que incorpore un sistema heterogéneo.

Partiendo de esta base, los desarrolladores de SkePU han desarrollado un selector de implementaciones adaptativo para su framework [20]. El argumento que sostiene el desarrollo de este selector es que distintas implementaciones proporcionan un rendimiento diferente en función del hardware en el que se ejecuten.

Para determinar qué implementación es más eficiente, utilizan “machine learning” en base a ejecuciones pasadas. En lugar de hacer un análisis exhaustivo probando todas las configuraciones para determinar la más adecuada, lo que llevaría un gran tiempo de ejecución, optan por usar una búsqueda jerárquica en base a heurísticas.

Las heurísticas se establecen determinando el rendimiento de las implementaciones en base a un subespacio de parámetros de contexto en los que se asume que si es la mejor implementación para estos parámetros también lo será para el conjunto de ellos. Por ejemplo, para un espacio dimensional de un parámetro, se comprueba el rendimiento en base a dos valores “a”, “b” y se asume que, si una implementación proporciona los mejores tiempos de ejecución para estos valores, también lo hará para todos los valores entre “a” y “b”.

De esta forma, SkePU propone una solución interesante pero limitada por el hecho que no utiliza en tiempo de ejecución la información que se puede extraer de la ejecución definitiva, ya que solo utiliza el histórico de ejecuciones de prueba.

2.8. METODOLOGÍAS ÁGILES

Por metodologías ágiles se entienden aquellos conjuntos de principios que aplicados de forma consistente en la realización de un proyecto permiten que los requisitos del mismo sean satisfechos eficientemente.

Estas metodologías buscan desglosar el proceso de desarrollo del proyecto en diferentes etapas o iteraciones, las cuales tienen asociadas unos objetivos concretos y un horizonte temporal. Al final de cada iteración, se realiza una reunión entre los miembros del equipo de trabajo para evaluar la consecución de dichos objetivos y plantear una discusión sobre el estado del producto desarrollado hasta el momento.

En este proceso iterativo, la participación del destinatario del proyecto es fundamental, ya que la consecución del proyecto de forma satisfactoria no se alcanzará sin la aprobación de este. Para alcanzar este fin, se emplearán ciertas técnicas para delimitar y definir las aspiraciones del destinatario o cliente de forma que se puedan materializar de forma consistente a lo largo del desarrollo.

Uno de los pilares fundamentales en del desarrollo ágil es la importancia individual de los miembros del equipo de trabajo, cuyas aportaciones personales han de ser recibidas por el resto de los miembros del equipo, discutidas y evaluadas. En esta línea de trabajo, la comunicación entre los miembros del equipo se incrementa y se obtiene un valor añadido que dista de otras formas de trabajo en las que un miembro del equipo es el único responsable de aportar ideas y el resto de miembros se limitan a seguir las directrices de este.

Esta forma de trabajo se consolida en el manifiesto ágil de 2001 [21] en el que se establecen los doce principios que resumen esta metodología de desarrollo:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.

8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

2.9. SCRUM

Dentro de las metodologías ágiles se encuentra Scrum, una filosofía de trabajo que se adapta a equipos reducidos y autosuficientes a los que se asigna un proyecto.

Este proyecto se desglosa en distintas actividades y objetivos que se acotan en el tiempo y a las que se asocia un responsable. Estos bloques temporales reciben el nombre de iteraciones o **sprints** y tienen una duración predeterminada que suele oscilar entre dos y cuatro semanas. Cada sprint va a suponer un desarrollo del proyecto que puede ser entregado al cliente si este lo solicita.

El siguiente esquema resume todos los elementos de Scrum.

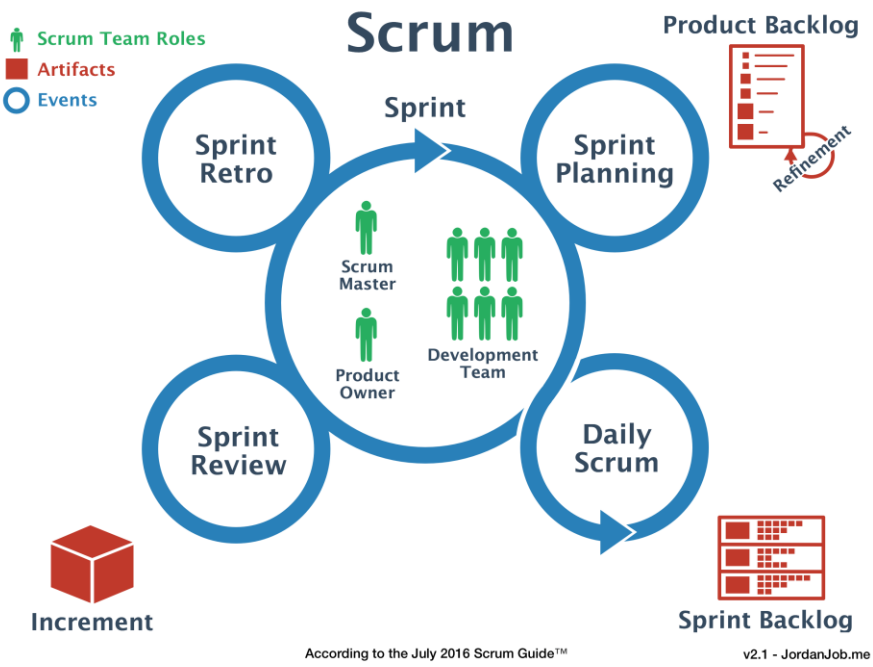


Figura 11: Diagrama Scrum. [37]

2.9.1. ROLES

Dentro de la metodología Scrum se definen tres roles:

1) Scrum master (facilitador)

El Scrum Master es el líder equipo de trabajo o director del proyecto.

Su función principal es organizar las etapas de Scrum y proporcionar los recursos necesarios para que los miembros del equipo puedan cumplir sus tareas en el tiempo estimado.

A su vez, es el vínculo principal de unión entre el cliente y el equipo de trabajo, por lo que será el encargado de transmitir a este último los requisitos que solicite y exponer los avances.

2) Product owner (cliente)

Es el actor o conjunto de actores que tienen una necesidad y solicita al equipo de trabajo una solución o producto para que sea satisfecha.

Deberá definir los requisitos principales que deberán cubrirse y será el encargado de realizar la evaluación final de los resultados.

Es deseable que participe de forma activa en el desarrollo del proyecto proporcionando nuevos requisitos, actualizando los ya fijados, estableciendo prioridades de los requisitos e intercambiando opiniones cada vez que se exponen los avances en el proyecto.

Es fundamental que no se introduzcan cambios en aquellos requisitos que se encuentran en un sprint activo, ya que desembocará en problemas de desarrollo y resultados insatisfactorios.

3) Team (equipo de trabajo)

El equipo de trabajo es el conjunto de individuos que desarrollarán el proyecto partiendo de una serie de requisitos y transformándolos en un producto que cubra las necesidades del product owner.

En la metodología Scrum los equipos no son muy extensos, preferiblemente entre cinco y nueve personas. Esta limitación de tamaño se justifica en la premisa de que un número elevado de miembros facilita que la comunicación y colaboración se debilite, llegándose a formar subgrupos de trabajo que no estarán en plena coordinación entre ellos.

Los equipos de trabajo deben tener un carácter multidisciplinar para poder abordar todas las tareas derivadas de los requisitos sin depender de agentes externos al equipo.

En este contexto se recomienda que todo el equipo de trabajo se encuentre en la misma localización física para favorecer la comunicación y generar un ambiente de trabajo productivo.

2.9.2. PLANIFICACIÓN

En cada sprint se realiza una disección de los requisitos clave junto al cliente, esta primera etapa es conocida como **planificación**, y en ella se resuelven todas las dudas al respecto, de forma que se obtenga de forma satisfactoria una lista con los objetivos que se esperan lograr. A continuación, el equipo de trabajo aborda los requisitos y define una serie de tareas concretas que permitan la consecución de cada uno de ellos. Estas tareas son categorizadas según su prioridad y duración estimada y finalmente se asignan a cada miembro del equipo. Uno de los principales beneficios de realizar una planificación conjunta es la posibilidad de intercambiar ideas entre los miembros del equipo y tener claro qué está haciendo cada uno en cada momento, de esta forma la comunicación entre los responsables de tareas interdependiente será fluida y coordinada.

2.9.3. SPRINT

Una vez terminada la planificación cada miembro del equipo comienza a trabajar en las tareas asignadas en el **sprint**. A lo largo del mismo es conveniente que se realicen de forma regular reuniones de sincronización, generalmente diarias por lo que reciben el nombre de **Scrum daily meetings**. Estas reuniones permiten exponer cualquier inconveniente que no se previó en la planificación y que podría afectar a las tareas del resto de miembros del equipo. La figura del facilitador o Scrum Master toma en este momento gran importancia ya que tratará de proporcionar a los miembros del equipo las herramientas necesarias para que las tareas asignadas se cumplan en el plazo previsto. La duración de estas reuniones debe ser breve, aproximadamente treinta minutos.

2.9.4. RETROSPECTIVA

Una vez acabado el plazo del sprint se lleva a cabo una revisión, **Sprint Retrospective**, donde se analizan los resultados obtenidos, los problemas que han surgido detallando sus causas y se proponen soluciones para evitar o reducir estas dificultades en el próximo sprint.

2.9.5. BENEFICIOS

El beneficio clave que se obtiene utilizando la metodología Scrum viene dado de su carácter iterativo y consiste en la flexibilidad en el desarrollo del proyecto y la comunicación entre miembros del equipo y el cliente final. Es habitual que el cliente, o **product owner**, que solicita el proyecto y define los requisitos básicos, a lo largo del desarrollo cambie o introduzca nuevos requisitos debido a sus necesidades particulares. Esto supone un trastorno importante si se sigue una metodología de trabajo estática en la que todos los requisitos se definen en un primer momento y se va desarrollando el producto hasta que es presentado en su versión final. Gracias a la planificación en sprints, es factible introducir nuevos requisitos de forma cómoda y a su vez es posible mostrar a lo largo del desarrollo los avances al cliente para que este pueda tener una idea clara de cual está siendo el resultado de su producto. A su vez, en caso de que se cancelara el proyecto se contaría con un producto más o menos funcional dependiendo de cuantos sprints hayan sido terminados antes de la cancelación, ya que cada uno de estos debe ofrecer una mejora funcional al proyecto.

Finalmente se pueden resumir una serie de beneficios clave que aporta Scrum [22]:

1. Calidad: Scrum proporciona las herramientas necesarias para favorecer la mayor calidad del proyecto.
    a. Se definen y elaboran los requisitos de forma que las características del producto sean lo más precisas y relevantes posible.
    b. Se realiza una revisión periódica del proyecto en la que se involucra al cliente, de forma que los cambios se discuten cuando son recientes y todo el equipo de trabajo los tiene presente.
    c. Se realizan revisiones de los sprints de forma que el equipo de trabajo puede mejorar factores dependientes del equipo como el entorno de trabajo, las herramientas que se están empleando, las relaciones entre elementos, etc.

2. Reducción del "Time to market": Scrum favorece que el cliente reciba valor entre un treinta y un cuarenta por ciento antes que métodos tradicionales gracias a los siguientes factores:
    a. El desarrollo del proyecto comienza de forma más temprana que en metodologías tradicionales ya que se prescinde de tener una documentación completa antes de empezar el proyecto, la cual puede llevar meses definir. Esta información del producto se compensa con la involucración del cliente en el desarrollo.
    b. Se realiza una distinción entre requisitos de alta prioridad y requisitos de baja prioridad, de forma que antes de finalizar complemente el producto, se cuenta con una versión funcional que podría lanzarse al mercado mientras se desarrollan los requisitos que no son prioritarios.
    c. Cada sprint introduce nuevas características que van aportando valor al producto.

3. Incremento del retorno de la inversion o "Return Of Investment": Además de que el producto se pueda introducir en el mercado antes de tener todos sus requisitos acabados, hay otras razones por la que se aumenta el retorno de inversión.
    a. La revisión de los sprints con interacción de los stakeholders permite que se realicen correcciones en el proyecto de forma temprana, lo cual es más rápido y fácil de llevar a cabo que hacerlo cuando el producto está terminado.
    b. Se reducen los costes derivados de la corrección de errores gracias a que se realizan pruebas de funcionalidad en cada sprint, de esta forma los cambios se pueden introducir cuando los desarrolladores tienen presente esas funcionalidades en concreto y no afectan a subsiguientes funcionalidades que deriven de esta.

4. Aumento de la satisfacción del cliente:
    a. Se colabora directamente con el cliente o con un miembro de su equipo que tiene conocimientos técnicos sobre el producto que se está desarrollando.
    b. Se mantiene un backlog del proyecto actualizado y en el que se pueden introducir cambios que serán analizados y resueltos por el equipo.
    c. Se demuestra progresivamente el incremento de las funcionalidades del producto según se van terminando los sprints. Lo cual se deriva en que el cliente está recibiendo un producto funcional que se va mejorando periódicamente hasta que se finalice.
5. Moral del equipo de trabajo alta: Se favorece que todos los integrantes del equipo de trabajo estén satisfechos con su posición y tareas a desempeñar. Scrum favorece esto con:
    a. Los equipos de trabajo son reducidos, lo que permite una gran comunicación entre los integrantes del mismo y poder aportar ideas creativas y críticas constructivas.
    b. Los equipos de trabajo se pueden formar no solo en base a los conocimientos técnicos si no de la personalidad de los miembros.
    c. La integración de un product owner en el equipo reduce la separación entre organizaciones y favorece la comunicación entre ellas.
    d. El scrum master debe favorecer la integración de los miembros del equipo y prevenir cualquier conflicto que pueda surgir en el equipo e impedir interferencias externas.
    e. Trabajar en un equipo reducido con gran comunicación favorece que los miembros del equipo intercambien conocimientos y se reduzcan problemas a la hora de intercambiar ideas y confusiones.
6. Se aumenta la colaboración y la propiedad del proyecto: El equipo se encarga de un único proyecto de forma que se favorece la sensación de responsabilidad con el mismo lo cual ayuda a que se trate de aumentar todo lo posible su calidad. Esto queda plasmado en las siguientes características de Scrum:
    a. Trabajo conjunto entre el product owner, el scrum master y el equipo de desarrollo.
    b. Realización de los daily meetings, sprints y revisiones de los mismos de forma regular.
    c. Trabajar en el mismo espacio de trabajo
    d. Toma de decisiones de forma conjunta.

7. Métricas relevantes: las métricas que se utilizan en Scrum para estimar tiempo y constes son más precisas que las que se usan de forma tradicional por las siguientes razones:
    a. Las estimaciones de esfuerzo necesario para la realización de las tareas las llevan a cabo los propios desarrolladores.
    b. Los presupuestos y fechas límites se definen en base a las capacidades del equipo que está realizando el trabajo y no de forma abstracta.
    c. Gracias a los daily meetings es posible proporcionar información adicional a estas métricas como por ejemplo problemas o dificultades imprevistas.
    d. El final de cada sprint se puede hacer una valoración bien informada sobre el coste de desarrollo y el valor que este proporciona.
8. Mejora de la visibilidad de progreso: En los proyectos Scrum todos los actores pueden acceder al estado del proyecto en cualquier momento. Se favorece la transparencia lo que ayuda a identificar problemas, hacer predicciones, y ayudar al equipo de desarrollo en ciertos aspectos y más importante, a tiempo. Esto es gracias a las siguientes características:
    a. Favorecer la comunicación entre los miembros del equipo, stakeholders y scrum master.
    b. Realizar los daily meetings donde se intercambia información del proyecto.
    c. Utilizar la información de los daily meetings para definir los objetivos de los sprints.
    d. Realizar las retrospectivas de los sprints para identificar los factores que han favorecido al proyecto y los que no.
9. Reducción de riesgos: Scrum ayuda a mitigar el riesgo de fracaso del proyecto y prevenir así la perdida de grandes cantidades de tiempo y dinero. Para ello:
    a. Se desarrollan en un primer lugar los requisitos de alto riesgo y de mayor prioridad.
    b. Se hace un desarrollo en sprints por lo que es fácil hacer una valoración del producto en cada etapa.
    c. Tener un producto funcional desde un primer momento ayuda a que el cliente tenga una idea de si el producto cumple sus expectativas.

3. ANÁLISIS DEL SISTEMA

En esta sección se busca definir los objetivos principales que se deben de alcanzar para el desarrollo del sistema. Una vez identificados, se realizarán las historias de usuario sobre los elementos fundamentales que se deben definir para la consecución del desarrollo. Estas historias de usuario recogen las funcionalidades del sistema y por tanto los requisitos que satisfacen las exigencias del cliente. Están formadas por:

- Identificador: código que define inequívocamente la historia de usuario
- Nombre: debe ser breve, conciso y descriptivo
- Prioridad: Tiene tres valores, Alta, Media y Baja. A mayor prioridad más importante su desarrollo.
- Riesgo: Tiene tres valores, Alto, Medio, Bajo. Cuantifica el impacto que tiene sobre el proyecto un error en este componente.
- Descripción: Texto que define el objetivo de la historia de usuario. Se indicará en este apartado cualquier aspecto que deba ser considerado en su desarrollo.
- Validación: Recoge las condiciones que se deberán tener en cuenta para determinar la consecución exitosa de la historia de usuario una vez esté terminada.

Para su mejor compresión, las historias de usuario ser recogerán en tablas con el siguiente formato

Historia de usuario	
Identificador	HXX
Nombre	
Prioridad	
Riesgo	
Descripción	
Validación	

Tabla 1: Plantilla historia de usuario

3.1.1. OBJETIVOS

En esta sección se recogen los objetivos a alto nivel, o epics en nomenclatura Scrum, que se quieren alcanzar en este proyecto.

El objetivo principal es desarrollar un selector de implementaciones, es decir, un componente de software al que se le puedan definir distintas versiones para una determinada funcionalidad, de forma que este selector sea capaz de analizar un determinado problema y distinguir cuál de las diferentes versiones es la más eficiente para ese caso concreto.

Una vez definido el objetivo final, se pueden listar los elementos necesarios para su consecución:

1. Definición de diferentes implementaciones

   Debido a que el selector debe escoger entre diferentes versiones de una misma funcionalidad, se deben proporcional los mecanismos necesarios para la identificación de dichas versiones.

2. Definición del tamaño del problema

   Cada implementación debe tener sus propias características de forma que una predomine sobre otra dado un problema. De esta forma se debe poder establecer un tamaño de problema que ayude al selector a elegir una versión sobre otra.

3. Obtención de un perfil de ejecución:

   Es necesario poder analizar las ejecuciones de las diferentes versiones, de forma que el selector pueda tener esta información en cuenta a la hora de elegir entre versiones.

4. Almacenamiento y recuperación de perfiles de ejecución

   Los datos obtenidos deben poder guardarse de forma persistente de forma que esta información sea reutilizada por el selector.

5. Modelo de selección

   El selector debe tener definido el algoritmo en base al cual va a valorar las diferentes implementaciones para poder seleccionar la más adecuada en base a un problema.

3.1.2. HISTORIAS DE USUARIO

A partir de estos objetivos se pueden definir las distintas historias de usuario:

Historia de usuario	
Identificador	H01
Nombre	Identificación de implementaciones
Prioridad	Alta
Riesgo	Alto
Descripción	Se debe proporcionar un parámetro que se asigne a cada implementación de forma que el selector pueda identificarla.
Validación	El selector debe ser capaz de identificar y elegir diferentes implementaciones

Tabla 2: H01

Historia de usuario	
Identificador	H02
Nombre	Determinar tamaño del problema
Prioridad	Media
Riesgo	Medio
Descripción	Se debe proporcionar un parámetro que permita indicar el tamaño del problema en base al cual se elegirá la implementación para su resolución.
Validación	El selector debe ser capaz de obtener el tamaño del problema y usarlo para elegir entre diferentes implementaciones

Tabla 3: H02

Historia de usuario

Identificador	H03
Nombre	Obtener resultados de ejecución
Prioridad	Alta
Riesgo	Alto
Descripción	Debe ser posible obtener los tiempos de ejecución de cada implementación
Validación	El selector debe ser capaz de obtener el tiempo de ejecución de cada implementación y conservarlo en memoria

Tabla 4: H03

Historia de usuario

Identificador	H04
Nombre	Almacenar resultados de ejecución
Prioridad	Media
Riesgo	Bajo
Descripción	Los tiempos de ejecución deben poder guardarse de forma persistente
Validación	Los resultados de las ejecuciones se deben guardar en memoria secundaria

Tabla 5: H04

Historia de usuario

Identificador	H05
Nombre	Recuperar resultados de ejecución
Prioridad	Media
Riesgo	Bajo
Descripción	Los tiempos de ejecución deben poder guardarse de forma persistente
Validación	Los resultados de las ejecuciones se deben guardar en memoria secundaria

Tabla 6: H05

Historia de usuario	
Identificador	H06
Nombre	Definir modelo selección
Prioridad	Alta
Riesgo	Alto
Descripción	El selector debe seguir un modelo de predicción eficaz para seleccionar la mejor implementación respecto a cada problema
Validación	Se debe poder comprobar que el selector está siguiendo el modelo de selección definido

Tabla 7: H06

3.1.3. TAREAS

En esta sección se recogen las tareas específicas para llevar a cabo las historias de usuario. Estas tareas tendrán el siguiente formato:

- Identificador: código que define inequívocamente la tarea
- Historia de usuario: indica qué historia de usuario desarrolla
- Estado: indica el estado de la tarea. Los posibles valores son "No asignado", "En proceso", "Terminado". Cabe mencionar que toda tarea debe ser asignada a un miembro del equipo para que pueda realizarse.
- Iteración: Indica la iteración en la que se va a desarrollar
- Responsable: es la persona que está a cargo de la historia de usuario. Es posible que participen más miembros del equipo en la historia de usuario, pero estarán bajo las directrices del responsable.

Tarea	
Identificador	TXX
Historia de usuario	HXX
Responsable	
Iteración	
Estado	
Descripción	

Tabla 8: Plantilla tareas

Tarea	
Identificador	T01
Historia de usuario	H01
Responsable	Andrés Sánchez
Iteración	1
Estado	Completado
Descripción	Se debe proporcionar un comando de preprocesado (pragma) que permita identificar las diferentes implementaciones.

Tabla 9: T01

Tarea	
Identificador	T02
Historia de usuario	H01
Responsable	Andrés Sánchez
Iteración	5
Estado	Completado
Descripción	Debe de comprobarse que el selector identifica correctamente las diferentes implementaciones.

Tabla 10: T02

Tarea	
Identificador	T03
Historia de usuario	H01
Responsable	Andrés Sánchez
Iteración	3
Estado	Completado
Descripción	Debe poder indicarse el tamaño del problema a través del comando de preprocesado.

Tabla 11: T03

Tarea	
Identificador	T04
Historia de usuario	H06
Responsable	Andrés Sánchez
Iteración	4
Estado	Completado
Descripción	Debe comprobarse que los parámetros de las implementaciones son obtenidos correctamente por el modelo de selección

Tabla 12: T04

Tarea	
Identificador	T05
Historia de usuario	H02
Responsable	Andrés Sánchez
Iteración	5
Estado	Completado
Descripción	Debe comprobarse que el selector es capaz de obtener correctamente el tamaño del problema a través del comando de preprocesado.

Tabla 13: T05

Tarea	
Identificador	T06
Historia de usuario	H03
Responsable	Andrés Sánchez
Iteración	1
Estado	Completado
Descripción	El selector debe ser capaz de obtener el tiempo de ejecución de cada implementación.

Tabla 14: T06

Tarea	
Identificador	T07
Historia de usuario	H03
Responsable	Andrés Sánchez
Iteración	2
Estado	Completado
Descripción	Debe comprobarse que el selector obtiene tiempos de ejecución precisos.

Tabla 15: T07

Tarea	
Identificador	T08
Historia de usuario	H04
Responsable	Andrés Sánchez
Iteración	1
Estado	Completado
Descripción	El selector debe ser capaz de almacenar de forma persistente el tiempo de ejecución de cada implementación.

Tabla 16: T08

Tarea	
Identificador	T09
Historia de usuario	H04
Responsable	Andrés Sánchez
Iteración	2
Estado	Completado
Descripción	Debe comprobarse que la información obtenida de distintas ejecuciones se almacena de forma persistente.

Tabla 17: T09

Tarea	
Identificador	T10
Historia de usuario	H05
Responsable	Andrés Sánchez
Iteración	2
Estado	Completado
Descripción	Los resultados de ejecuciones anteriores deben de poder reutilizarse para selecciones posteriores.

Tabla 18: T10

Tarea	
Identificador	T11
Historia de usuario	H05
Responsable	Andrés Sánchez
Iteración	2
Estado	Completado
Descripción	Debe comprobarse que el selector carga correctamente los resultados de ejecuciones anteriores.

Tabla 19: T11

Tarea	
Identificador	T12
Historia de usuario	H06
Responsable	Javier Fernandez
Iteración	3
Estado	Completado
Descripción	Debe desarrollarse un modelo de selección que será implementado en el selector.

Tabla 20: T12

Tarea	
Identificador	T13
Historia de usuario	H05
Responsable	Andrés Sánchez
Iteración	4
Estado	Completado
Descripción	Debe comprobarse que el modelo de selección es aplicado correctamente por el selector.

Tabla 21: T13

3.1.4. SEGUIMIENTO DE TAREAS

A continuación, se muestra el panel de seguimiento de las tareas a realizar. Este panel recoge los siguientes campos:

- Identificador de la tarea: código único con el que se identifica cada tarea
- Iteración: Identificador numérico de la iteración en la que se realiza la tarea
- Prioridad: Campo de tres valores que indica la importancia de la tarea.
- Fecha de inicio de la iteración
- Fecha de vencimiento de la iteración
- % Completado: representa con un valor porcentual el estado de desarrollo en el que se encuentra una tarea.

Con este cuadro de seguimiento los miembros del equipo pueden ir representado el estado en el que dejan una tarea después de cada jornada para que se tenga constancia del estado del proyecto a lo largo de su desarrollo.

Seguimiento de Tareas

Identificador de Tarea	ITERACION	PRIORIDAD	FECHA DE INICIO	FECHA DE VENCIMIENTO	% COMPLETADO
T01	1	Alta	15/10/2017	28/10/2017	100%
T02	5	Alta	10/12/2017	23/12/2017	100%
T03	3	Alta	12/11/2017	25/11/2017	100%
T04	4	Media	26/11/2017	10/12/2017	100%
T05	5	Media	10/12/2017	23/12/2017	100%
T06	1	Alta	15/10/2017	28/10/2017	100%
T07	2	Alta	29/10/2017	11/11/2017	100%
T08	1	Media	15/10/2017	28/10/2017	100%
T09	2	Media	29/10/2017	11/11/2017	100%
T10	2	Media	29/10/2017	11/11/2017	100%
T11	2	Media	29/10/2017	11/11/2017	100%
T12	3	Alta	12/11/2017	25/11/2017	100%
T13	4	Alta	26/11/2017	10/12/2017	100%

Tabla 22: Seguimiento de tareas

4. DISEÑO DEL SISTEMA

El selector de implementaciones basado en probabilidades se va a articular sobre una serie de tecnologías que permitirán su funcionamiento. En este apartado se revisarán brevemente estas tecnologías y su contribución al selector de implementaciones, además de mostrarse cuál será el flujo de trabajo de selector.

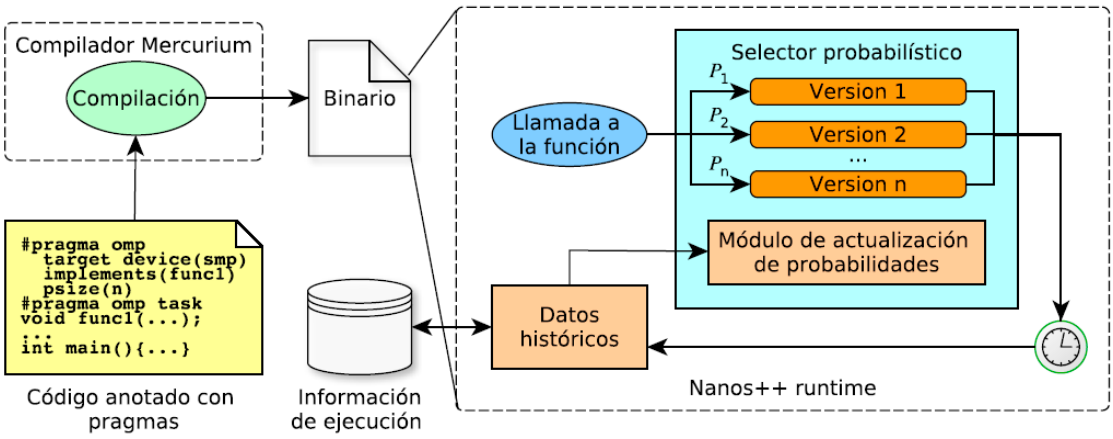


Figura 12: Esquema del framework

4.1. LENGUAJE DE PROGRAMACIÓN

Cómo se mencionó anteriormente, el framework OmpSs da soporte a tres lenguajes de programación, C, C++ y Fortran. Ya que el objetivo es desarrollar el selector de implementaciones bajo este framework, se escogió el lenguaje C++ para su implementación.

Ya que el lenguaje Fortran escapa de los conocimientos del autor, la elección de lenguajes se reduce a C y C++, por lo que es conveniente exponer cuales son las razones de la elección de C++ sobre C.

C++ es un lenguaje de programación que, a pesar de tener una larga trayectoria, principios de 1980 [23], es uno de los lenguajes más extendidos y que cuenta con una gran comunidad de desarrolladores y usuarios, ejemplo de ello es la fundación Standard C++ Fundation [24], una organización sin ánimo de lucro destinada a desarrollar y promover el uso del lenguaje C++. Esta organización publica nuevas funcionalidades y características al lenguaje, en el momento de la redacción de este documento la última revisión publicada es el standard C++17 y la versión C++20 se encuentra en desarrollo. A su vez, las tecnologías que deben ser modificadas para la introducción del selector se basan en C++, por lo que el uso de este lenguaje permite mantener una coherencia entre el selector y las herramientas de las que hace uso.

4.2. SELECCIÓN BASADA EN PROBABILIDADES

El algoritmo de selección de probabilidades se basa en atribuir una probabilidad a cada implementación de una determinada función. Esta probabilidad se basa en el tamaño del problema y resultados de ejecuciones previas.

Al contar con distintos tamaños de problema, es necesario hacer una distinción de las probabilidades en base a este factor, para ello, se dividen los posibles rangos de tamaño en intervalos de idéntica longitud, de esta forma se asignará una probabilidad distinta a cada implementación de cada intervalo. Por ejemplo, si tuviéramos dos implementaciones y dos posibles tamaños de problema, tendríamos dos intervalos de tamaño y para cada intervalo dos probabilidades distintas debido a que hay dos implementaciones.

Una vez que se extrae el tamaño del problema, el algoritmo debe elegir una de las implementaciones, para ello seguirá la estrategia de la ruleta. Esta aproximación consiste en dividir cada intervalo en una serie de subintervalos, uno por cada implementación, cuyo tamaño viene definido por la probabilidad que dicha implementación tiene asignada. A continuación, se define un número pseudo aleatorio entre cero y uno que indicará qué implementación se va a aplicar en base al segmento seleccionado.

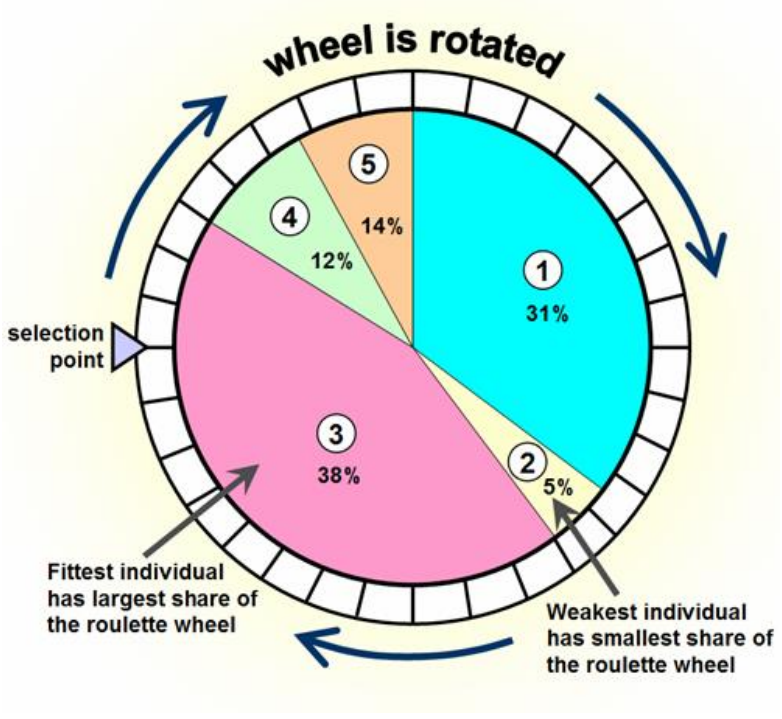


Figura 13: Esquema estrategia de la ruleta. [38]

4.3. CLÁUSULA PSIZE

A la hora de evaluar las distintas implementaciones de una funcionalidad, uno de los parámetros más importantes es el tamaño del problema. Como se ha mencionado anteriormente. este valor va a utilizarse para asignar una probabilidad a cada implementación junto con la información que se tiene almacenada de ejecuciones anteriores.

El tamaño del problema deberá ser introducido por el usuario y para ello es necesario proporcionar un mecanismo con este fin. Gracias a la utilización del framework OmpSs es posible incluir una nueva cláusula, que será denominada "psize", la cual indicará la posición del argumento que contiene el tamaño del problema en la función que se está analizando.

Como se puede observar, la palabra "psize" es una abreviatura de las palabras "problem size" que significan tamaño del problema en inglés.

A continuación, se muestra un ejemplo de cuál sería la implementación en código:

```
# pragma omp target device ( smp ) psize (2)
# pragma omp task
void func ( int **m, int problemSize );
```

Figura 14: Ejemplo cláusula psize

La cláusula "psize" aparece junto a la directiva target, que indica al compilador que debe crear una tarea que se asignará a un dispositivo en el que se ejecutará el bloque de código que le acompaña. El valor de la cláusula "psize" es "2" lo que informa de que será el segundo argumento de la función el que contenga el tamaño del problema. En este ejemplo, el segundo argumento recibe el nombre de "problemSize".

La inclusión de esta cláusula supuso la modificación de la directiva "target" en el compilador Mercurium.

La siguiente figura muestra un ejemplo del código de una aplicación que estaría usando tanto las cláusulas de OmpSs como la cláusula "psize".

```
#pragma omp task out(v0) label(myFunction)
void myFunction(){/*...*/}

#pragma omp target device (Device1) implements(myFunction)
#pragma omp task out(v0) label(myFunction_version1)
void myFunctionAlternativeImplementation(){/*...*/}

# pragma omp target device ( smp ) psize (2)
# pragma omp task
void func ( int **m, int problemSize );
```

Figura 15: Ejemplo código aplicación

4.4. MERCURIUM

Mercurium es el compilador source-to-source del framework de OmpSs y gracias a este es posible transformar los pragmas que aporta OmpSs en llamadas que puedan ser interpretadas por el runtime Nanos.

El uso de este compilador favorece al desarrollo de aplicaciones al permitir escribir un código reducido que será transformado en otro más complejo y así ahorrar tiempo en la programación y depuración de código además de aportar legibilidad al mismo. Otro aspecto interesante es su accesibilidad a la hora de ser extendido por parte de terceros para aportar funcionalidades adicionales. En este caso, se pretende modificar para introducir un nuevo pragma, proceso de modificación que se desarrollará en la sección de implementación.

Otra de las virtudes de Mercurium es que permite gestionar código para diferentes dispositivos, aspecto importante a la hora de desarrollar para plataformas heterogéneas, y da soporte al uso de diferentes compiladores, como por ejemplo el compilador *nvcc* de Nvidia para la compilación de código destinado a GPUs de esta empresa. Con este objetivo, Mercurium es capaz de reestructurar el código e incluso generar código nuevo, los archivos y binarios adicionales serán integrados en un único archivo que contendrá la información del resto, la finalidad es seguir la estructura de que por cada archivo de entrada se genere un único archivo de salida.

El flujo de trabajo de Mercurium es el siguiente:

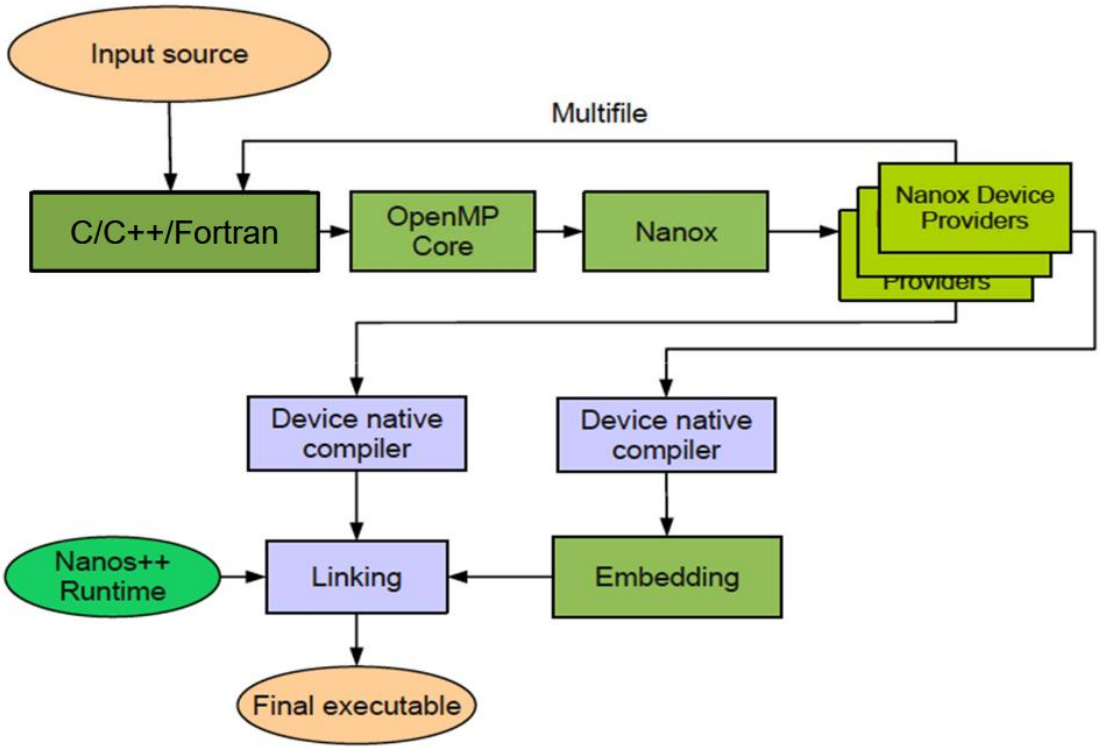


Figura 16: Flujo de trabajo Mercurium [39]

Como se puede observar en el esquema de trabajo de Mercurium, este acepta un archivo de entrada que podrá estar desarrollado en los lenguajes C, C++ o Fortran y partiendo de este se extraerán diferentes archivos si los dispositivos de salida lo requieren, los cuales serán compilados por los compiladores correspondientes y finalmente se empaquetarán en un único archivo el cual será compilado para generar un único archivo ejecutable.

Esta forma de trabajo proporciona una gran flexibilidad al desarrollador al contar de forma centralizada con un código fuente que puede contener tareas para dispositivos de diferente naturaleza. Este hecho lo convierte en una opción interesante para el desarrollo del selector de implementaciones.

4.5. NANOS++

Nanos es el runtime de OmpSs y destaca por ofrecer una serie de características interesantes, entre ellas destacan las siguientes:

- Soporte para tres lenguajes de programación, C, C++ y Fortran
- Funcionalidades para dispositivos específicos
- Soporte para la creación y manipulación de tareas
- Proporciona una serie de planificadores de tareas
- Gestión de memoria
- Compatibilidad con el programa Extrae para la instrumentalización
- Diseño modular que permite la integración de nuevos componentes

La siguiente imagen muestra la arquitectura de Nanos:

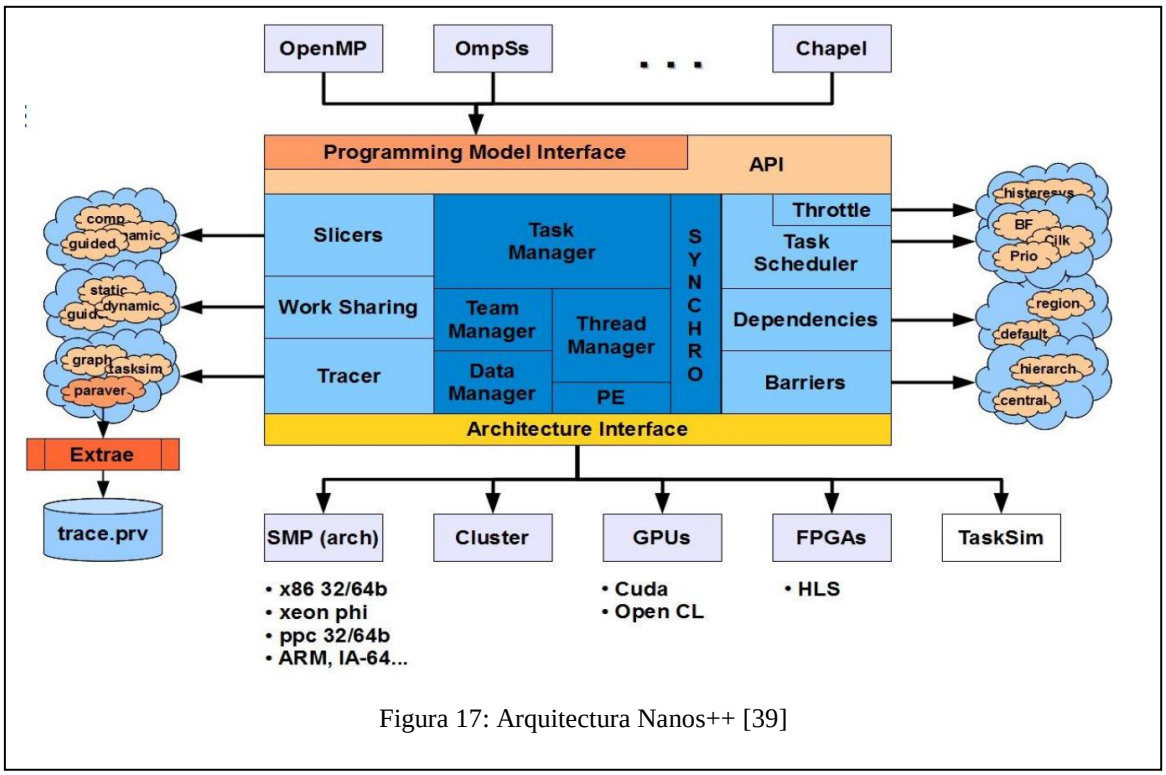


Figura 17: Arquitectura Nanos++ [39]

El esquema anterior muestra cómo Nanos puede recibir información de diferentes interfaces como OmpSs y OpenMP y a partir de ellas utiliza el sistema de gestión y sincronización de tareas que pueden emplear diferentes planificadores para su administración y ser ejecutadas e instrumentalizadas. Estas tareas pueden ser paralelizadas haciendo uso del grafo de dependencias que puede ser generado mediante clausulas específicas por el usuario definidas en el código fuente enviado a Mercurium.

Las arquitecturas soportadas por Nanos, tal y como se muestran en la figura, son variadas, cabe destacar SMP siendo CPUs de propósito general, GPU compatibles con CUDA y OpenCL, sistemas de cluster e incluso puede utilizarse en arquitecturas simuladas (tasksim).

De forma complementaria, el diseño de Nanos permite la introducción de nuevos componentes, entre los que destaca la introducción de planificadores adiciones, factor que es fundamental en el presente trabajo ya que el selector de implementaciones va a integrarse dentro de Nanos y hacer uso de los recursos que este proporciona para la gestión de tareas.

4.6. ALGORITMO DE ACTUALIZACIÓN DE PROBABILIDADES

La primera cuestión es determinar qué factor va a ser el que determine un mejor o peor rendimiento. En este caso se ha optado por partir del tiempo de ejecución, de forma que la implementación que proporcione un menor tiempo de ejecución será considerada como la que aporta un mejor rendimiento.

De esta forma se va a seguir la ecuación 1, expuesta a continuación, para determinar si una implementación tiene un rendimiento superior a las implementaciones que se han analizado anteriormente y de las que se tiene información.

$$\text{Best(A; S)} = \forall i \in S: E(A) \leq E(i)$$

Ecuación 1:Selección implementación rendimiento superior

Esta ecuación indica que partiendo de un conjunto de implementaciones S, la implementación A tiene un rendimiento mayor al conjunto S, si su tiempo de ejecución E(A), es menor al tiempo de ejecución de todas las implementaciones anteriores.

El cálculo de las probabilidades se basa en emplear la media y la desviación típica sobre el tiempo de ejecución de cada implementación para extraer el intervalo de confianza de cada versión. El siguiente paso es hacer una comparación entre los intervalos de confianza y obtener así su probabilidad.

En caso de que dos intervalos sean disjuntos, la implementación que tiene el menor tiempo de ejecución y por tanto el mayor rendimiento, tendrá una probabilidad del cien por cien de ser seleccionada. Sin embargo, en caso de que los intervalos se solapen, la probabilidad será repartida entre las distintas versiones. Mientras más ejecuciones se obtengan, los intervalos de confianza se verán reducidos hasta que las implementaciones tengan intervalos disjuntos. A la hora de realizar el cálculo de las probabilidades, se debe tener en cuenta que el valor esperado se debe de contener en el intervalo de confianza y se va a distribuir de forma homogénea en dicho intervalo.

A continuación, se muestra un ejemplo del cálculo de la probabilidad para tres implementaciones:

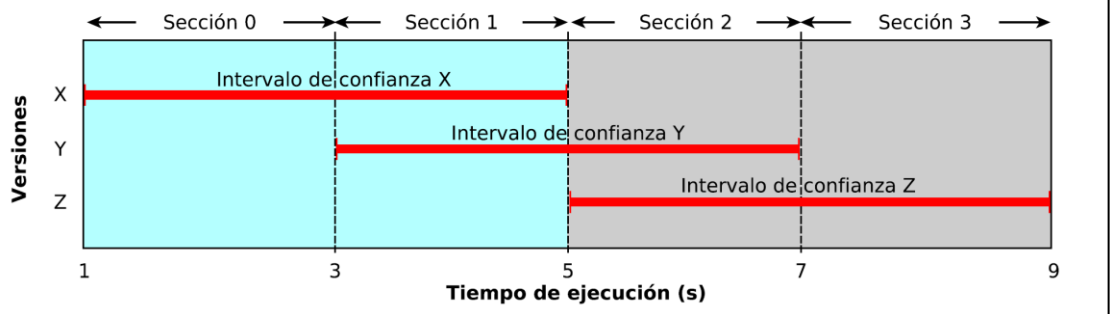


Figura 18: Ejemplo cálculo probabilidades

En este ejemplo se cuenta con tres implementaciones denominadas X, Y Z. En la figura anterior se puede observar que cada implementación cuenta con un intervalo de confianza que se extiende a lo largo del eje X que representa el tiempo de ejecución.

Como se puede apreciar en la figura, los intervalos no son disjuntos, si no que se solapan para ciertos valores del tiempo de ejecución. Las distintas secciones que se muestran vienen definidas por los valores en los que cada intervalo comienza y termina, de esta forma se cuenta con cuatro secciones numeradas de cero a tres.

Debido a que la implementación X muestra el intervalo de confianza que comienza en el menor tiempo de ejecución, solo se considerarán las secciones que contienen a este intervalo de confianza. Esto se debe a que aquellas secciones posteriores incluyen implementaciones que no pueden proporcionar una mejor solución, ya que su tiempo de ejecución es mayor al peor caso posible de la implementación X, por lo que esta tendría un rendimiento superior. En el presente ejemplo, solo se considerarán las dos primeras secciones, dejando de lado las dos últimas.

Una vez definidas las secciones relevantes, se empleará el teorema de la probabilidad total para obtener las probabilidades de cada implementación acumulando las probabilidades marginales de las secciones relevantes. En el ejemplo, se puede observar que la implementación Z no se encuentra dentro de las secciones relevantes por lo que el selector no escogerá esta versión en ningún caso.

A la hora de obtener las probabilidades marginales de cada implementación en una sección concreta, es necesario aplicar por segunda vez el teorema de la probabilidad total para realizar la comparación de cada implementación con el resto de las implementaciones. Este algoritmo se deberá repetir una vez por cada implementación contenida en la sección.

Se calcularán tres probabilidades marginales ya que se deben de obtener las probabilidades cuando el tiempo de ejecución esperado de la alternativa está contenido dentro de la sección seleccionada, cuando su valor esperado es menor que el extremo inferior de la sección y cuando su valor esperado es mayor al extremo superior de la sección.

Las siguientes ecuaciones muestran respectivamente el cálculo de la probabilidad de las implementaciones X, Y y Z. La variable I_n representa la sección n, CI_j representa el intervalo de confianza para la implementación j, y B_t y E_t representan el principio y el final de una sección o intervalo de confianza t respectivamente.

$$\mathbf{P}\big(Best(X,\{X,Y,Z\})\big) = \mathbf{P}\big(Best(X,\emptyset) \mid \mathbb{E}(X) \in I_0\big)\mathbf{P}\big(\mathbb{E}(X) \in I_0\big) +$$
$$\mathbf{P}\big(Best(X,\{Y\}) \mid \mathbb{E}(X) \in I_1, \mathbb{E}(Y) \in I_1\big)\mathbf{P}\big(\mathbb{E}(X) \in I_1\big)\mathbf{P}\big(\mathbb{E}(Y) \in I_1\big) +$$
$$\mathbf{P}\big(Best(X,\{Y\}) \mid \mathbb{E}(X) \in I_1, \mathbb{E}(Y) > I_1\big)\mathbf{P}\big(\mathbb{E}(X) \in I_1\big)\mathbf{P}\big(\mathbb{E}(Y) > I_1\big) =$$
$$1 \cdot \frac{E_{I_0} - B_{I_0}}{E_{CI_X} - B_{CI_X}} + \frac{1}{2} \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_X} - B_{CI_X}} \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_Y} - B_{CI_Y}} + 1 \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_X} - B_{CI_X}} \cdot \frac{E_{CI_Y} - E_{I_1}}{E_{CI_Y} - B_{CI_Y}} =$$
$$1 \cdot \frac{3-1}{5-1} + \frac{1}{2} \cdot \frac{5-3}{5-1} \cdot \frac{5-3}{7-3} + 1 \cdot \frac{5-3}{5-1} \cdot \frac{7-5}{7-3} = \frac{28}{32} = 0{,}875$$

Ecuación 2: Cálculo probabilidad implementación X

$$\mathbf{P}\big(Best(Y,\{X,Y,Z\})\big) =$$
$$\mathbf{P}\big(Best(Y,\{X\}) \mid \mathbb{E}(Y) \in I_1, \mathbb{E}(X) \in I_1\big)\mathbf{P}\big(\mathbb{E}(Y) \in I_1\big)\mathbf{P}\big(\mathbb{E}(X) \in I_1\big) =$$
$$\frac{1}{2} \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_Y} - B_{CI_Y}} \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_X} - B_{CI_X}} = \frac{1}{2} \cdot \frac{5-3}{7-3} \cdot \frac{5-3}{5-1} = \frac{4}{32} = 0{,}125$$

Ecuación 3: Cálculo probabilidad implementación Y

$$\mathbf{P}\big(Best(Z,\{X,Y,Z\})\big) = \mathbf{P}\big(Best(Y,\emptyset) \mid \mathbb{E}(Z) \in \emptyset\big) = 0$$

Ecuación 4: Cálculo probabilidad implementación Z

5. IMPLEMENTACIÓN Y EVALUACIÓN

5.1. IMPLEMENTACIÓN

En esta sección se va a explicar qué pasos se ha de seguir para incluir en el framework de OmpSs un nuevo planificador.

El primer paso es determinar en qué directorio es preciso guardar el archivo que contenga en nuevo planificador. Debido a que el planificador se emplea en tiempo de ejecución, el sistema encargado de utilizarlo dentro del framework de OmSs es el runtime Nanos++.

En los archivos de Nanos++ es posible encontrar una carpeta denominada "plugins", dentro de la cual se encuentra la carpeta destinada a almacenar todos los planificadores, esta carpeta recibe el nombre de "sched". Por lo tanto, el archivo que contenga al planificador deberá localizarse dentro de los plugins de Nanox, en la siguiente ruta:

- "*nanox-XXX\src\plugins\sched\probabilisticSelector.cpp*"

El siguiente paso consiste en determinar qué elementos son necesarios definir en el nuevo planificador para que sea correctamente reconocido por el runtime Nanos++.

El planificador deberá incluir una serie de headers como son:

- schedule.hpp
- wddeque.hpp
- plugin.hpp
- system.hpp

Una vez definidos los headers de Nanos++ e introducidos los que sean necesarios para el funcionamiento del planificador, es preciso definir dos espacios de nombres o "namespace" en los que se debe incluir todo el código del planificador. El primer "namespace" recibe el nombre de "nanos" y el segundo, incluido dentro del primero, recibe el nombre de "ext".

```
namespace nanos {
        […]
        namespace ext {[…]}
}
```

Figura 19: Namespace para Nanos++

A continuación, se deberá crear una clase base para el planificador que deberá extender la clase "SchedulePolicy" donde se definirán los métodos públicos y privados que sean necesarios para el planificador desarrollado.

```
class ProbabilisticSelector : public SchedulePolicy {[…]}
```

Figura 20: Clase del planificador en Nanos++

Una vez definida la lógica del planificador y los elementos que este requiera para su correcto funcionamiento, el último paso en la creación del planificador es definir su nombre para el framework, para ello, fuera del namespace "nano" es preciso declarar el planificador de la siguiente manera:

```
DECLARE_PLUGIN("placeholder-name", nanos::ext::ProbabilisticSelector);
```

Figura 21: Declaración de plugin en Nanos++

Una vez definido el planificador, se deberá actualizar el archivo makefile dentro de la carpeta "*nanox-XXX\src\plugins\sched*" para incluir el nuevo planificador.

```
scheduler_sources=\

   sched/scheduler.cpp \

   $(END)

[…]

debug_LTLIBRARIES +=\

 debug/libnanox-sched-scheduler.la

debug_libnanox_sched_scheduler_la_CPPFLAGS=$(common_debug_CPPFLAGS)

debug_libnanox_sched_scheduler_la_CXXFLAGS=$(common_debug_CXXFLAGS)

debug_libnanox_sched_scheduler_la_LDFLAGS=$(AM_LDFLAGS)
$(ld_plugin_flags)

debug_libnanox_sched_scheduler_la_SOURCES=$(scheduler_sources)

[…]

instrumentation_debug_LTLIBRARIES +=\

 instrumentation-debug/libnanox-sched-scheduler.la

instrumentation_debug_libnanox_sched_scheduler_la_CPPFLAGS=$(common_in
strumentation_debug_CPPFLAGS)

instrumentation_debug_libnanox_sched_scheduler_la_CXXFLAGS=$(common_in
strumentation_debug_CXXFLAGS)

instrumentation_debug_libnanox_sched_scheduler_la_LDFLAGS=$(AM_LDFLAGS
) $(ld_plugin_flags)

instrumentation_debug_libnanox_sched_scheduler_la_SOURCES=$(scheduler_
sources)

[…]

performance_LTLIBRARIES +=\

 performance/libnanox-sched-scheduler.la

performance_debug_libnanox_sched_scheduler_la_CPPFLAGS=$(common_instru
mentation_debug_CPPFLAGS)

performance_debug_libnanox_sched_scheduler_la_CXXFLAGS=$(common_instru
mentation_debug_CXXFLAGS)

performance_debug_libnanox_sched_scheduler_la_LDFLAGS=$(AM_LDFLAGS)
$(ld_plugin_flags)

performance_debug_libnanox_sched_scheduler_la_SOURCES=$(scheduler_sour
ces)
```

Figura 22: Modificación makefile para incluir nuevo planificador

5.2. MODIFICACIÓN DE MERCURIUM

Debido a que Mercurium no cuenta de forma nativa con pragma para obtener el tamaño del problema, ha sido preciso modificarlo para incorporar este nuevo pragma.

Para la modificación del compilador fue preciso modificar los siguientes archivos:

./src/tl/omp/core/tl-ompss-target.hpp

./src/tl/omp/core/tl-ompss.hpp

./src/tl/omp/core/tl-ompss.cpp

./src/tl/omp/nanox-nodecl/devices/cuda/nanox-cuda.cpp

./src/tl/omp/nanox-nodecl/devices/smp/nanox-smp.cpp

./src/tl/omp/nanox-nodecl/tl-devices.hpp

./src/tl/omp/nanox-nodecl/tl-lower-task-call.cpp

./src/tl/omp/nanox-nodecl/tl-lower-task.cpp

./src/tl/omp/nanox-nodecl/tl-lowering-visitor.hpp

Figura 23: Archivos Modificados Mercurium

En estos archivos se define el nombre del pragma, qué tipo de información va a recibir y qué valores se consideran válidos para el pragma.

En este caso, el tamaño del problema viene definido por un número entero, por lo que se guarda como una variable “integer” en el compilador y su nombre es “psize”.

El siguiente ejemplo muestra un fragmento de código en C++ en el que se define una función y el tamaño del problema y a continuación se muestra cuál es el archivo de salida tras pasar por el compilar Mercurium.

```
#pragma omp target device (smp) psize (2)
#pragma omp task
void ph1(int &x, int y){
    x = x+y;
}
```

Figura 24: Ejemplo pragma psize previa compilación

Esta función llamada “ph1” va a ser ejecutada en el dispositivo “smp” y tiene un tamaño de problema con valor 2.

Una vez procesado por Mercurium se obtiene lo siguiente:

```
[…]

static ::nanos_smp_args_t smp_ol_ph1_args = { (void (*)(void *))(void
(*)(::nanos_args_0_t &))::smp_ol_ph1,  "ph1",  2};

[…]
```

Figura 25: Ejemplo pragma psize compilado

Se puede observar como Mercurium recoge como segundo parámetro el tamaño del problema que se ha definido. Esta información será interpretada por el runtime Nanos++ y utilizada por el planificador.

5.3. MODIFICACIÓN DE NANOS++

Una vez que el compilador source-to-source es capaz de interpretar el nuevo pragma, que determina el tamaño del problema, y convertirlo en código que el runtime pueda entender, es preciso modificar el runtime para que acepte este nuevo parámetro y lo utilice en las tareas.

Para ello es preciso modificar una serie de archivos dentro del runtime:

./src/apis/c/nanos_wd.cpp

./src/arch/smp/smpdd.hpp

./src/core/nanos-int.h

./src/core/workdescriptor.hpp

./src/core/workdescriptor_decl.hpp

Figura 26: Archivos modificados Nanos++

Dos de los elementos más importantes del runtime respecto al planificador son el elemento “device” o dispositivo y el elemento “work descriptor” o unidad de trabajo.

Los objetos “device” van a contener toda la información del dispositivo como su nombre, identificador, funciones para reservar y librar memoria… Estos elementos se ven acompañados de los datos de dispositivo o “device data”, donde se va a almacenar la información específica a cada dispositivo. Respecto al planificador, nos interesa añadir dos campos a este objeto, el nombre de la función que se encuentra en el dispositivo y el tamaño del problema asignado a esta función.

Para estar seguros de contar con variables que puedan almacenar correctamente el nombre de función y tamaño de problema, se escogió un vector de caracteres de mil posiciones para el nombre de la función y una variable de numero entero “integer” para el tamaño del problema, de la misma forma que se hizo en el compilador.

```
typedef struct {
    void (*outline) (void *);
    char func_name[1000];
    unsigned int problem_size;
} nanos_smp_args_t;
```

Figura 27: Estructura de Nanos++ para tamaño de problema

Como se puede observar en la figura anterior, el nombre de esta es el que va a utilizar Mercurium cuando procese el pragma y el parámetro que definen el tamaño del problema, tal y como se puede observar en la Figura 26: Archivos modificados Nanos++.

Respecto a los “work descriptor”, hay que indicar que estos elementos conforman las unidades de trabajo que el planificador va a gestionar. De estas unidades de trabajo va a ser posible extraer los tiempos de ejecución de cada implementación que se utilizan por el planificador para determinar cuál es la más eficiente siguiendo esta métrica.

5.4. VERIFICACIÓN Y VALIDACIÓN

En este apartado se recogen las pruebas llevadas a cabo para asegurar el correcto funcionamiento del planificador. Para ello se va a describir una serie de escenarios teóricos que se pondrán en práctica y se compararán los resultados que se esperan obtener con los realmente obtenidos.

5.4.1. ENTORNO DE PRUEBA

En este apartado se enumeran los componentes del entorno en el que se realizan las pruebas de verificación y validación. En este caso, el entorno de pruebas difiere considerablemente del entorno de evaluación, debido a que las pruebas de este apartado no requieren una gran capacidad de cálculo. A su vez, utilizar el planificador en distintas máquinas nos permite garantizar hasta cierto punto la portabilidad del selector.

- Intel Core i7-6700K de cuatro núcleos a 4GHz
- 16GB de memoria RAM DDR4
- Una GPU Nvidia GTX 1070
- Sistema operativo Linux Ubuntu 16.04.6 LTS kernel v4.4

5.4.2. CRITERIO DE ACEPTACIÓN

Las pruebas de esta sección pueden realizarse en cualquier orden, ya que son independientes entre sí. Estas pruebas se definen expresamente para comprobar que la selección es certera, por lo que no se busca plantear casos de uso prácticos, ya que estos se realizarán en el apartado de evaluación, en este caso.

Una prueba se considerará superada cuando la selección por parte del selector del dispositivo donde ejecutar la función de prueba sea el esperado de forma teórica.

5.4.3. ESPECIFICACIÓN DE LOS CASOS DE PRUEBA

A continuación, se detalla la plantilla según la que se definen las pruebas, los elementos que la definen y por último los casos de prueba realizados.

Identificador	**CPXX**
Descripción	
Resultado Esperado	

Tabla 23: Plantilla prueba verificación

- Identificador: Código alfanumérico precedido por los caracteres CP (caso de prueba) y completado con dos dígitos que identifican inequívocamente la prueba.
- Descripción: fragmento de texto que explica la prueba
- Resultado Esperado: recoge el funcionamiento esperado del selector

Identificador	**CP01**
Descripción	Se define una función en la que se reciben dos valores y se devuelve la suma de ambos. En la implementación del procesador se introduce además un bucle en el que la suma se realiza 1000 veces antes de devolver el valor.
Resultado Esperado	El selector debe escoger la implementación de la GPU

Tabla 24: CP01

Identificador	**CP02**
Descripción	Se define una función en la que se reciben dos valores y se devuelve la suma de ambos. En la implementación de la GPU se introduce además un bucle en el que la suma se realiza 1000 veces antes de devolver el valor.
Resultado Esperado	El selector debe escoger la implementación del procesador

Tabla 25: CP02

Identificador	**CP03**
Descripción	Se define una función en la que se va a realizar la suma de los valores de un vector. Se crean dos implementaciones para CPU, la implementación A va a consistir en un bucle en el que se suman los valores iterativamente. La implementación B va a consistir en usar el patrón "reduce" para hacer la suma. Se realiza una ejecución con valores de 1 a 10. Se realiza una segunda ejecución con valores del 1 al 1000000.
Resultado Esperado	En la primera ejecución el selector optará por la implementación A. En la segunda ejecución el selector optará por la implementación B.

Tabla 26: CP03

Identificador	**CP04**
Descripción	Se define una función en la que se va a realizar la suma de los valores de un vector. Se crean dos implementaciones para CPU, la implementación A va a consistir en un bucle en el que se suman los valores iterativamente. La implementación B va a consistir en usar el patrón "reduce" para hacer la suma. En esta prueba se va a ir incrementando progresivamente el tamaño del vector que se envía a la función. Se comienza con un vector de una posición con valor 1 y se termina con un vector de 1000000 posiciones con valores del 1 al 1000000.
Resultado Esperado	El selector debe optar por la implementación A inicialmente, pero deberá cambiar a la implementación B según el tamaño del vector crece

Tabla 27: CP04

Identificador	**CP05**
Descripción	Se define una función en la que se reciben dos valores y se devuelve la suma de ambos. De esta función se realizan dos implementaciones, la A en la que se realiza este comportamiento sin modificaciones y la implementación B en la que se introduce además un bucle en el que la suma se realiza 1000 veces antes de devolver el valor. Se realiza un entrenamiento de 10 ejecuciones y se termina la ejecución. A continuación, se realiza una nueva ejecución.
Resultado Esperado	El selector debe optar por la implementación A desde la primera iteración haciendo uso del histórico.

Tabla 28: CP05

Identificador	**CP06**
Descripción	Se define un vector de 1000 elementos con valores numéricos enteros aleatorios del 1 al 100 y un vector de 1000 elementos. Se crea una función que debe elevar al cuadrado cada elemento del primer vector y guardar el valor resultante el segundo vector en la posición correspondiente al primer vector. Se crean dos implementaciones para CPU, la implementación A va a consistir en un bucle que obtiene el valor de una posición del vector, la eleva al cuadrado y la almacena en el segundo vector antes de pasar al siguiente elemento. La implementación B va a consistir en realizar el mismo proceso, pero con cuatro hilos de forma paralela.
Resultado Esperado	El selector deberá optar por la implementación B.

Tabla 29: CP06

Identificador	**CP07**
Descripción	Se define una función en la que se crean dos vectores de 1000 elementos, se rellena con valores aleatorios del 1 al 100 y se crea un tercer vector del mismo tamaño. En la implementación A se hace la suma de los elementos de los dos vectores y se almacena en un tercer vector. En la implementación B se obtiene el valor del primer vector y se eleva al valor del segundo vector en la posición correspondiente y se almacena en un tercer vector.
Resultado Esperado	El selector debe optar por la implementación A

Tabla 30: CP07

5.5. EVALUACIÓN DE RESULTADOS

En esta sección se exponen los dos métodos que se han utilizado a la hora de comprobar la eficacia del planificador basado en probabilidades. Por una parte, se realizará un ejercicio de multiplicación de matrices, General Matrix Multiplication (GeMM), para evaluar tanto el rendimiento como la precisión del planificador. El siguiente método consiste en realizar una comparativa con el selector propuesto por OmpSs denominado *versioning*.

5.5.1. ESPECIFICACIONES DEL ENTORNO DE EVALUACIÓN

Las pruebas llevadas a cabo para la evaluación del planificador se han realizado en una de las máquinas del laboratorio de ARCOS de la Universidad Carlos III de Madrid. Esta cuanta con las siguientes especificaciones:

- Dos procesadores Intel Xeon Ivy Bridge E5-2695 v2 los cuales proporcionan 24 núcleos a 2.40 GHz de frecuencia.
- Caché L3 de 30MB
- 128GB de memoria RAM tipo DDR3
- Una GPU AMD Radeon modelo R90X
- Una GPU AMD R9 modelo 285
- Un coprocesador Intel Xeon Phi 3120
- Sistema operativo Linux Ubuntu 14.04.2 LTS con kernel v3.13.0-57

A su vez, del framework OmpSs se ha utilizado el compilador Mercurium en su versión 2.0 y el runtime Nanos++ en su versión 0.12a.

El código fuente generado por el compilador Mercurium se compila utilizando el compilador GCC versión 5.1 y la opción de optimización -O3 que activa todas las opciones de optimización proporcionadas por este compilador.

5.5.2. MULTIPLICACIÓN DE MATRICES

La prueba de rendimiento utilizando la multiplicación de matrices se ha llevado a cabo utilizando las librerías clBLAS e Intel MKL [25]. Es importante remarcar que la librería clBLAS hace uso de todos los procesadores de la máquina utilizada para este ejercicio, mientras que la librería Intel MKL solo utiliza el procesador Intel Xeon.

De esta forma, se cuenta con cinco implementaciones diferentes entre las cuales el selector puede elegir:

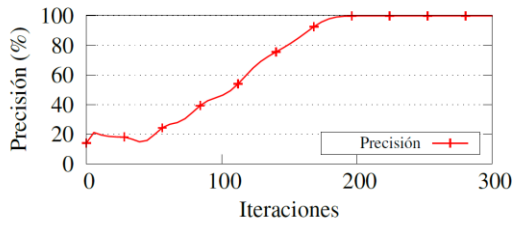
- Implementación en Intel MKL
- Implementación de clBLAS en procesador Intel Xeon Ivy Bridge
- Implementación de clBLAS en coprocesador Intel Xeon Phi
- Implementación de clBLAS en AMD Radeon modelo R90X
- Implementación de clBLAS en AMD R9 modelo 285

Esta prueba de concepto evalúa tanto el rendimiento del selector como su precisión. El rendimiento se calcula en base al tiempo de ejecución comparado entre la implementación seleccionada por el planificador y la implementación con menor tiempo de ejecución, de esta forma el rendimiento sería la división del tiempo de ejecución de la implementación más rápida entre la implementación seleccionada por el planificador. La precisión por su parte consiste en la división del número de veces que el planificador acierta al seleccionar la versión correcta entre el número de iteraciones.

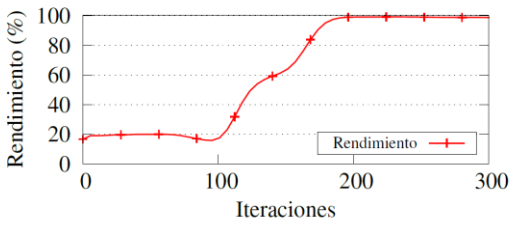
El ejercicio consiste en realizar una serie de iteraciones, en cada cual, se va a entrenar al selector realizando una multiplicación de matrices cuadradas cuyo tamaño cambia en cada iteración tomando valores aleatorios entre 64x64 y 4096x4096. En cada iteración, el seleccionador deberá elegir una implementación que permita la resolución de la multiplicación en el menor tiempo posible.

Para saber cuál es la implementación que proporciona menor tiempo de ejecución a la hora de realizar la multiplicación de matrices de tamaño aleatorio, se realizan cien ejecuciones de dicha multiplicación de matrices sobre cada implementación. A continuación, se lleva a cabo la evaluación de los resultados obtenidos en el entrenamiento.

Las siguientes figuras representan estos resultados:



(a) Precisión en iteraciones de entrenamiento.



(b) Rendimiento en iteraciones de entrenamiento.

Figura 28: Resultados multiplicación matrices

Es posible observar en las gráficas anteriores como la precisión se ve incrementada, a medida que aumenta el número de iteraciones realizadas, hasta alcanzar una precisión de 99.8% aproximadamente, tras la realización de ciento setenta iteraciones. Los intervalos de confianza se ven reducidos hasta permitir al planificador seleccionar la implementación más adecuada en cada caso.

La gráfica respecto al rendimiento ofrece un resultado similar al de la precisión, sin embargo, cabe destacar que cada vez que el planificador se equivoca al escoger la implementación adecuada, el impacto que tiene en el rendimiento es mayor que en la precisión.

5.5.3. COMPARATIVA DE PLANIFICADORES

En esta sección se expone los resultados obtenidos tras comparar el planificador basado en probabilidades desarrollado en este trabajo con el planificador *versioning* ofrecido por OmpSs.

En este caso, la prueba que se va a realizar consiste en un *benchmark* sintético en el que se ejecutan dos bucles de treinta iteraciones. En el primer bucle se realizará la multiplicación de matrices cuadradas con tamaño fijo de 1024x1024. El segundo bucle consistirá en la realización de una multiplicación de matrices cuadradas cuyo tamaño variará aleatoriamente entre 64x64 y 4096x4096, cabe entender que en cada iteración se emplean los dos planificadores por lo que a pesar de que sea aleatorio el tamaño de las matrices a multiplicar, ambos planificadores tendrán el mismo caso para cada iteración.

Los resultados obtenidos en el *benchmark* son los siguientes:

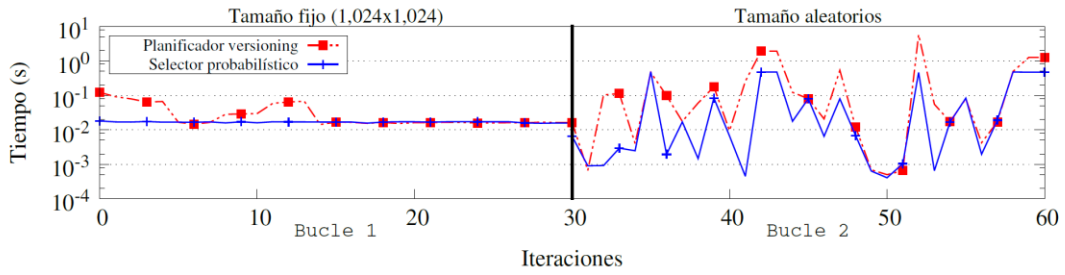


Figura 29: Resultados benchmark

Respecto a los resultados sobre el primer bucle, con tamaños fijos de matrices, se puede apreciar que el selector probabilístico desde la primera ejecución selecciona la implementación con menor tiempo de ejecución, esto es debido a que el planificador cuenta con información sobre ejecuciones previas, en este caso las trescientas iteraciones que se realizaron en el caso de uso anterior. Por su parte, el seleccionador *versioning* no cuenta con ningún mecanismo que le permita almacenar información de ejecuciones anteriores, por lo que necesita ejecutar las distintas implementaciones hasta encontrar aquella que ofrece un mejor rendimiento.

Otro aspecto importante a tener en cuenta es el hecho de que el planificador basado en probabilidades utiliza el tamaño del problema a la hora de realizar la sección, por lo que a diferentes tamaños de problema puede variar la elección de implementaciones. El planificador *versioning* en cambio, no cuenta con esta funcionalidad, por lo que, aunque cambie el tamaño del problema, una vez que ha seleccionado cual es la implementación que considera más eficiente, no escogerá una implementación diferente, aunque proporcione un mejor resultado.

De esta forma, el planificador basado en probabilidades ofrece dos funcionalidades, histórico de ejecuciones y tamaño de problema, que le permite ofrecer mejores resultados frente al planificador *versioning*.

6. PLANIFICACIÓN Y ENTORNO SOCIOECONÓMICO

Esta sección recoge la planificación pormenorizada del proyecto, se realiza una estimación sobre los costes asociados al mismo y se describe cuál podría ser el impacto podría suponer en el contexto socioeconómico.

6.1. PLANIFICACIÓN TEMPORAL

En este apartado se describe la planificación que se ha llevado a cabo en el desarrollo del proyecto.

El proyecto se ha llevado a cabo en dos meses y medio aproximadamente, desde el 2 de octubre de 2017 hasta el 23 de diciembre de 2017.

Las tareas que conforman el proyecto son las siguientes:

- Definición del proyecto: esta fase consiste en la conceptualización del proyecto, donde se determina en qué va a consistir de forma general y en su aceptación por parte de la universidad.

  En este caso, se definió que se quería desarrollar un planificador con la capacidad de elegir entre diferentes versiones de una misma función.

- Análisis del proyecto: en esta tarea se identifican concretamente los elementos que van a componer el proyecto y sus características.

  Esto se traduce en la concreción de los objetivos a alcanzar en el proyecto y en la extracción a partir de estos de las historias de usuario a través de las cuales se obtendrán las tareas concretas a realizar.

  Este proceso se traduce en:

  - Definición de objetivos
  - Definición de historias de usuario
  - Definición de tareas a realizar
  - Definición de pruebas de verificación y validación
  - Definición de pruebas de evaluación

- Iteraciones SCRUM: siguiendo la metodología ágil SCRUM se busca identificar las iteraciones que van a conformar el proyecto, su duración y las tareas concretas que van a incluirse en cada iteración. Estas son:
    - Iteración 1: Se llevan a cabo las tareas relacionadas con la definición de los mecanismos para obtener información tanto de las diferentes implementaciones como de las métricas de rendimiento.
    - Iteración 2: Se comprueba la calidad de las métricas y el correcto almacenaje y recuperación de información.
    - Iteración 3: Se introducen los mecanismos para determinar el tamaño del problema y se plantea el modelo de selección.
    - Iteración 4: Se realizan comprobaciones sobre la correcta obtención de los parámetros de implementaciones y la aplicación adecuada del modelo de selección.
    - Iteración 5: Se verifica y valida el funcionamiento del selector.
- Evaluación: En esta tarea se llevan a cabo las pruebas de evaluación del selector y la comparativa con los selectores ya existentes.
- Documentación: se lleva a cabo la redacción del presente documento.

6.2. DIAGRAMA DE GANTT

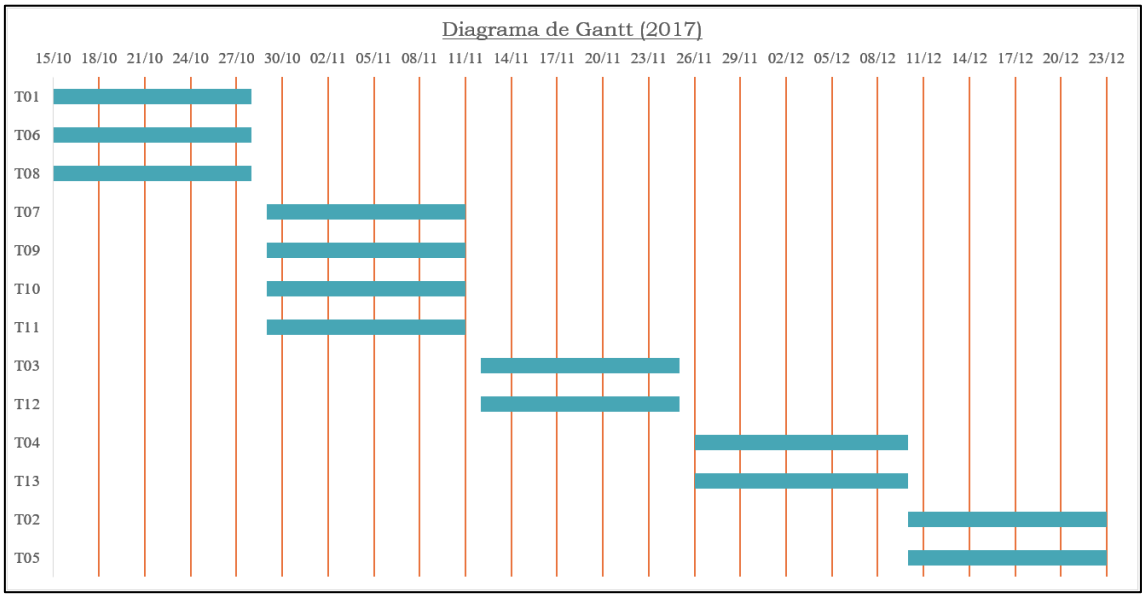


Figura 30: Diagrama de Gantt

6.3. PRESUPUESTO

En esta sección se hace un desglose del coste derivado del desarrollo del proyecto. Como se ha expuesto anteriormente, la duración del proyecto se ha prolongado durante dos meses y medio aproximadamente.

6.3.1. COSTE DE PERSONAL

El proyecto ha sido llevado a cabo principalmente por una única persona que ha desarrollado distintos roles dentro del proyecto, por lo que en función del rol hay un coste por hora diferente.

Respecto al número de horas trabajadas, se han trabajado cuatro horas diarias, de lunes a viernes dentro del plazo de realización del proyecto. Debido a que la fecha de inicio del proyecto es el 2 de octubre y la fecha de finalización es el 23 de diciembre, han transcurrido 84 días de los cuales 60 han sido trabajados. Por lo tanto, se tiene un total de 240 horas trabajadas.

Personal

Apellidos, Nombre	Categoría	Horas trabajadas	Coste por hora (€)	Coste total (€)
Sánchez Cuadrado, Andrés	Diseñador	20	7	140
Sánchez Cuadrado, Andrés	Analista	20	7	140
Sánchez Cuadrado, Andrés	Desarrollador	150	6	900
Sánchez Cuadrado, Andrés	Tester	50	5	250
			Total	1430

Tabla 31: Costes Personal

6.3.2. COSTES DE HARDWARE

Por coste de hardware se entienden todos los elementos físicos o materiales que han sido utilizados en el desarrollo del proyecto.

Para el desarrollo se han utilizado dos equipos, el equipo de desarrollo y el equipo de evaluación. El equipo de desarrollo es un ordenador de sobremesa que durante el desarrollo del proyecto se ha destinado íntegramente a este fin. Del mismo modo, el equipo de evaluación durante este periodo no se empleó en otra tarea que no estuviera relacionada con este proyecto. No obstante, el uso de la máquina de evaluación se limita a un mes, ya que las primeras pruebas no se empezaron a ejecutar hasta que el desarrollo del proyecto se encontraba en una fase avanzada.

Equipos

Descripción	Coste (€)	% dedicado al proyecto	Dedicación (meses)	Periodo de deprecación (meses)	Coste imputable[1]
Ordenador de sobremesa	500	100	2	36	27,28
Máquina utilizada en la evaluación	3900	100	1	60	65
				Total	92,28

Tabla 32: Costes Hardware

[1] Fórmula de cálculo de la amortización

$$\frac{\text{Nº meses desde la fecha de facturación} \times \text{Coste del equipo sin IVA} \times \text{\% de uso que se dedica al proyecto}}{\text{Periodo de deprecación}}$$

6.3.3. COSTE DE SOFTWARE

Los costes recogidos en esta sección se refieren a aquellos relacionados con los programas utilizados en el desarrollo del proyecto y del presente documento.

Software

Descripción	Coste por unidad (€)	Cantidad	Total
Ubuntu 14.04.2 LTS	0	1	0
Ubuntu 16.04.6 LTS	0	1	0
Framework OmpSs	0	1	0
Sublime Text 2	0	1	0
LXR Cross Referencer	0	1	0
Gnuplot	0	1	0
Microsoft Office 360 (versión de educación)	0	1	0
		Total	0

Tabla 33: Costes Software

6.3.4. OTROS COSTES DIRECTOS

Este apartado recoge los costes asociados a los materiales de oficina empleados en el desarrollo del proyecto, así como aquellos asociados a la utilización de los equipos utilizados en el mismo.

Otros costes directos

Descripción	**Coste imputable**
Material fungible oficina	15
Material fungible informático	10
Electricidad	20
Total	45

Tabla 34: Otros costes

6.3.5. PRESUPUESTO FINAL

A continuación, se indica el coste total del proyecto y se agrega el Impuesto sobre el Valor Añadido (IVA) vigente en España. Debido a que es un desarrollo sin ánimo de lucro, no se incluirán factores como el riesgo y el beneficio.

Resumen de costes (en euros)

Descripción	Valor
Coste de personal	1430
Coste de hardware	92,28
Coste de software	0
Costes de funcionamiento	0
Costes directos	45
Total sin IVA	1567,28
Total (21% IVA incluido)	1896,41

Tabla 35: Presupuesto final

6.4. IMPACTO SOCIAL DEL PROYECTO

En esta sección se va a discutir brevemente las repercusiones que podrían acontecer, en el contexto económico, social y medioambiental, si el proyecto expuesto en este trabajo se implantara de forma generalizada.

6.4.1. SITUACIÓN ACTUAL

En el momento de la redacción de este documento, se estima que cerca de la mitad de los hogares a nivel mundial cuentan con al menos un ordenador.

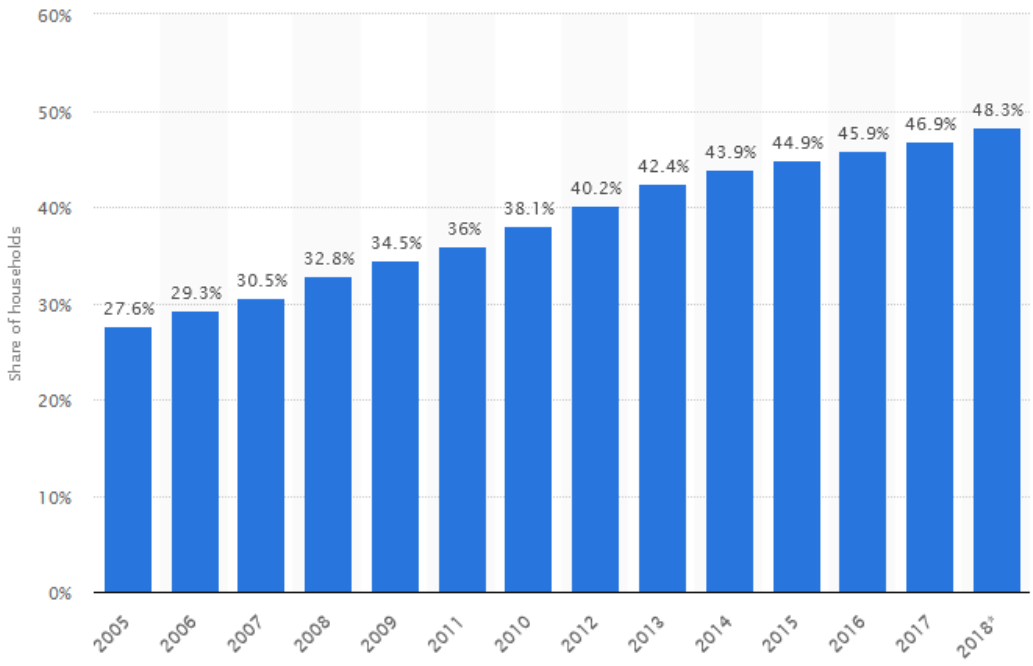


Figura 31: Ordenadores por vivienda 2005-2018 [40]

A su vez, según datos de la asociación GSMA, organización relacionada con compañías y operadores móviles, se estima que aproximadamente el 67% de la población mundial cuenta con un teléfono inteligente.

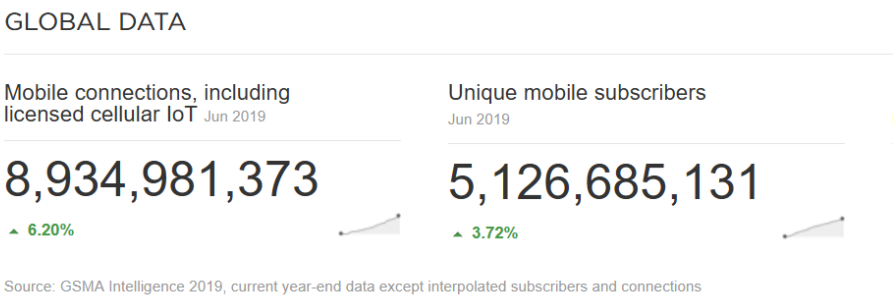


Figura 32: GSMA teléfonos por persona [41]

Estos datos son relevantes respecto al proyecto debido a que tanto los teléfonos inteligentes como los ordenadores personales pueden ser considerados plataformas homogéneas o heterogéneas y son susceptibles de contar con programas que hagan uso de los distintos dispositivos de procesamiento. Por tanto, el selector de implementaciones no ve teóricamente limitada su adopción a equipos destinados a la investigación o gran cantidad de cómputo, si no que podría llegar a implementarse en estos dispositivos teniendo cierta repercusión, la cual se va a tratar de definir en las siguientes secciones.

6.4.2. IMPACTO ECONÓMICO

Como se ha visto en las figuras de la sección anterior, el mercado de los ordenadores y dispositivos móviles se encuentra en un estado de crecimiento. Esto implica que cada vez es mayor la necesidad de abordar problemas mediante estos dispositivos teniendo en cuenta las capacidades y limitaciones de los mismo, ya sean vinculados al ámbito del entretenimiento, de la seguridad, operaciones ligadas a transacciones económicas, intercambio de datos y un sinfín de aplicaciones.

Esto implica que se requiere hasta cierto punto el aprovechamiento de los recursos que contienen estos dispositivos, por lo que la incorporación del selector de implementaciones puede favorecer dicha gestión de recursos al adaptar las tareas concretas a los elementos de cada terminal priorizando diferentes objetivos. Actualmente, el selector de implementaciones propuesto busca mejor la eficiencia en términos de tiempo de ejecución, lo cual implica que un dispositivo puede ver su vida útil alargada al ser utilizado de forma eficiente.

La prolongación de la vida útil de los dispositivos puede afectar al marco económico de forma tanto favorable como perjudicial. Sobre los beneficios cabría destacar el hecho de que los desarrolladores de programas ven su mercado ampliado, al haber más dispositivos sobre los que pueden lanzar su producto. En cambio, los fabricantes de dispositivos pueden ver su demanda reducida debido a que los clientes no necesitan cambiar de dispositivo con la frecuencia que lo hacen actualmente, viéndose esta reducida.

Por otra parte, haciendo hincapié en el hecho de que el selector de implementaciones desarrollado busca reducir el tiempo de ejecución, se puede extrapolar un beneficio asociado a aquellos sectores en los que esto sea un factor relevante a la hora de obtener un beneficio económico. Uno de los posibles escenarios sería el mercado financiero, en el cual la velocidad a la que se realizan las operaciones afecta directamente a los beneficios que se pueden obtener, debido a la constante fluctuación de precios sobre distintos elementos como pueden ser acciones de empresas. Otro sector es el de la manufactura en el que distintos programas digitales estén involucrados, de esta forma, llevar a cabo cálculos clave a la hora de la fabricación de productos puede llegar a aumentar la producción y con ello tener un impacto significativo en el sector económico.

En definitiva, resulta complicado estimar el efecto real de la implantación del selector de forma generalizada, al igual que valorar la negatividad o positividad del mismo. No obstante, es posible afirmar que el uso extendido de estos programas tendría un impacto que merecería ser estudiado en profundidad por expertos.

6.4.3. IMPACTO SOCIAL Y MEDIOAMBIENTAL

De la mano de las posibles repercusiones económicas, vienen las consecuencias que el uso del selector de implementaciones puede causar en el ámbito social y medioambiental.

Manteniendo la finalidad de minimizar el tiempo de ejecución de distintos programas, el selector de implementaciones puede afectar a sectores como el de la manufactura y por tanto tener repercusiones a nivel social, favoreciendo o empeorando las condiciones laborales de los trabajadores según se utilice las ventajas que aporta el uso del selector. Por ejemplo, si reducir el tiempo de ejecución de las tareas informáticas conlleva el poder aumentar la producción manual de productos y con ello generar más puestos de trabajo, estaríamos ante una situación beneficiosa en términos de empleabilidad. Por el contrario, si la mejora de rendimiento se produce sobre elementos automáticos de la producción, es posible que se integren más elementos de esta naturaleza al proceso de manufactura y por lo tanto se podrían ver reducidos los puestos de trabajo.

De forma similar se puede ver afectado el medioambiente, pues una mayor producción puede implicar una mayor generación de deshechos o por el contrario pueden verse favorecidos procedimientos vinculados al tratamiento de estos.

No obstante, si el selector de implementaciones basado en probabilidades expuesto en este trabajo se viera modificado para priorizar la eficiencia en términos de uso de energía, se podría llegar a ver una reducción significativa en términos de uso de energía y por tanto de utilización de los recursos para generarla.

A forma de conclusión sobre esta sección, destacar que el hecho de reducir los tiempos de ejecución de distintos programas o la priorización del uso eficiente de la energía, son conceptos extremadamente generales como para hacer una valoración objetiva fundamentada sobre el impacto que puede llegar a tener la adopción sistemática de esta tecnología. Se ha querido por tanto hacer una primera reflexión sobre el tema partiendo de la premisa que el selector de implementaciones basado en probabilidades fuese integrado de forma masiva, pero es preciso recalcar que el uso del selector no supone a priori un factor determinante en el ámbito socioeconómico actual.

7. MARCO REGULADOR

En esta sección se va a analizar cuál es la legislación vinculante al producto desarrollado en este trabajo, en caso de que exista y se abordarán las cuestiones relacionadas con la propiedad intelectual y el código ético y deontológico.

7.1. LEGISLACIÓN

Debido a que existe una gran cantidad de material legislativo que puede afectar a un producto informático, se va a tratar de exponer aquellos más significativos y la medida en la que afectan al presente trabajo.

Antes de abordar caso por caso, es preciso indicar que el selector de implementaciones es una herramienta que se puede integrar en proyectos y soluciones por parte de terceros, por lo que el uso que se haga en el sistema donde se integre el selector quedará a responsabilidad del desarrollador o desarrolladores a su cargo. En esta sección solo se van a mencionar los aspectos vinculantes con el presente proyecto y la información utilizada tanto en su desarrollo como evaluación.

- Protección de datos personales

En base al artículo 18.4 de la Constitución española, la ley debe limitar el uso de la informática para garantizar el honor y la intimidad de los ciudadanos y el ejercicio de sus derechos. De esta forma, se define el Código de Protección de Datos de Carácter Personal cuya última modificación data del 19 de marzo de 2019 en el momento de redactar el presente documento.

Siguiendo dicho código, los datos personales recogidos por un programa informático deben de haber sido obtenidos con el consentimiento del usuario, deben ser exactos y se debe de ofrecer ciertas garantías para mantener su privacidad.

En este caso, el selector de implementaciones hace uso de la información del dispositivo sobre el que se ejecuta, concretamente sobre los elementos de procesamiento del mismo. Esta información es proporcionada por el usuario a la hora de definir las distintas implementaciones. No obstante, la información no es distribuida y no se almacenan la información concreta sobre los dispositivos, ya que se hace referencia a la implementación y no sobre qué dispositivo se ejecuta.

- Comunicaciones y contenidos y medios audiovisuales

En base a lo dispuesto en el artículo 18.4 de la constitución anteriormente mencionado, se crea el Código de las Telecomunicaciones, cuya última modificación es del 15 de enero de 2015. En este código se recogen las garantías que deben cumplirse en la distribución y almacenamiento de las comunicaciones entre ciudadanos mientras que el Código de Derecho Audiovisual con última modificación de 22 de mayo de 2019, recoge lo propio sobre materiales audiovisuales que puedan afectar a las libertades y derechos de los ciudadanos.

En este caso, el selector de implementaciones no hace uso de ningún medio audio visual ni de ningún mecanismo de comunicación o almacenamiento de estos, que afecte a alguno de los bienes protegidos por los anteriores códigos.

7.2. CÓDIGO ÉTICO Y DEONTOLÓGICO

En lo que respecta a la ingeniería informática, el 19 de junio de 2015 se publico el Real Decreto 518/2015 por el que se aprueban los estatutos generales de los colegios oficiales de ingeniería en informática y de su consejo general.

En este se indica que debe definirse un código deontológico único para toda la profesión y el consejo general deberá garantizar la aplicación del mismo. En el caso del colegio de ingenieros informáticos de la Comunidad de Madrid, se recogen unas reglas de conducta entre las que cabe mencionar la responsabilidad, veracidad e integridad y honradez del profesional informático [26].

En lo que respecta al presente trabajo, destacar que pese a no formar parte de ningún colegio profesional de ingenieros informáticos, se han respetado los principios generales de estos.

7.3. PROPIEDAD INTELECTUAL

En lo que respecta al proyecto, este se va a poner a disposición del repositorio institucional de la Universidad Carlos III de Madrid, siendo posible acceder al mismo de forma libre y gratuita, respetando las condiciones de uso de la licencia Creative Commons, concretamente las condiciones recogidas en la modalidad de "Reconocimiento-uso no comercial sin obra derivada" [27] de forma que es posible compartir y distribuir el contenido de este documento, es posible crear un contenido nuevo a partir de la información que se encuentra en este siempre y cuando se reconozca la autoría del creador del documento original, no se utilice el contenido del documento con finalidades comerciales y en caso de que se cree un documento a partir de la información recogida en este, la obra resultante deberá ser difundida bajo la misma licencia.

8. CONCLUSIONES

En esta sección se va a revisar hasta qué punto se han cumplido los objetivos establecidos al comienzo del trabajo. A continuación, se expondrá brevemente varias percepciones personales sobre el trabajo y finalmente se darán algunos matices sobre cómo se podría desarrollar el proyecto en futuras revisiones.

8.1. CONCLUSIONES DEL PROYECTO

El planificador basado en probabilidades que se ha presentado en este trabajo supone un avance respecto a los planificadores de referencia en entornos heterogéneos y se plantea como una herramienta que puede utilizarse de forma intuitiva en un proyecto en el que sea preciso dar una respuesta eficiente, en términos de tiempo de ejecución, en un entorno en el que los problemas sean cambiantes y se requiera una adaptación a necesidades dinámicas.

Llegados a este punto, es preciso valorar si se han cumplido los objetivos que fijaron al comienzo del proyecto y en qué medida. Para ello se enumerarán a continuación los objetivos y se discutirá sobre qué se ha hecho para cumplirlos.

1. Debe ser capaz de identificar diferentes implementaciones de una misma función

Para alcanzar este objetivo, se han empleado los pragmas como mecanismo para identificar la implementación base y las diferentes alternativas. De esta forma, el planificador puede reconocer y emplear todas las alternativas a una determinada función y poner en marcha el algoritmo basado en probabilidades para hacer una selección eficaz.

2. Debe ser capaz de seleccionar de forma dinámica la implementación más eficiente en función del problema a resolver

Este objetivo se ha alcanzado utilizando el algoritmo de selección basado en probabilidades. Utilizando este algoritmo, se evalúan en tiempo de ejecución las distintas implementaciones y se escoge la más adecuada para el problema a resolver.

3. Debe ser capaz de almacenar la información obtenida en ejecuciones anteriores para llevar a cabo la selección de implementación

Tal y como se muestra en el esquema del “framework”, se cuenta con un archivo, en este caso de texto, que almacena la información de ejecuciones anteriores y es reutilizado en ejecuciones posteriores.

4. Debe proporcionar al usuario una interfaz intuitiva para su uso

Respecto a este punto, se considera que el uso de las cláusulas “implements”, “target device” y “psize” proporcionan una forma sencilla de utilizar la intuitiva, ya que con estos tres elementos el desarrollador puede especificar, qué implementaciones corresponden qué función base, sobre qué dispositivo se va a ejecutar y finalmente, en la implementación base se puede establecer el tamaño del problema para el que se debe de utilizar.

Se cuenta así con una herramienta que no solo cumple los objetivos, si no que proporciona un rendimiento superior al del planificador versioning, proponiendo una alternativa a considerar en el desarrollo de proyectos sobre plataformas heterogéneas.

8.2. REPERCUSIÓN

Como resultado de este proyecto, se publicó un artículo en uno de los workshop internaciones de la conferencia ICA3PP de 2017 la cual se enfocó en algoritmos y arquitecturas para el procesamiento en paralelo [28]. ICA3PP o International Conference on Algorithms and Architectures for Parallel Processing son una serie de conferencias que llevan realizándose desde 1995 y se centran en dar a conocer los avances en las áreas de algoritmos y arquitecturas para la computación en paralelo. Al tener un carácter internacional, el lugar de convocatoria cambia en cada edición, incluyendo Granada en 2016 y en el caso de la conferencia que incluye este documento, tuvo lugar en Finlandia. [29]

A su vez, se publicó en la conferencia nacional Jornadas SARTECO de 2017, concretamente en las Jornadas de Paralelismo [30]. SARTECO o la Sociedad de Arquitectura y Tecnología de Computadores, es una asociación sin ánimo de lucro que desde 2006 pretende favorecer el desarrollo científico y tecnológico en España. Su actividad más relevante son las Jornadas SARTECO donde se promueven los avances en las áreas de arquitectura y computadores mediante ponencias en las que se exponen los últimos artículos de investigación. [31]

8.3. OPINIÓN PERSONAL

A título personal considero que la realización de este proyecto en el departamento de ARCOS de la Universidad Carlos III de Madrid, me ha proporcionado una experiencia muy interesante al comprobar de primera mano cómo se lleva a cabo un proyecto de investigación.

Además, me ha proporcionado una oportunidad para profundizar en el estudio de herramientas complejas de software, como es un selector de implementaciones y familiarizarme más con herramientas basadas en directivas como OpenMP, las cuales se estudian en el programa de la universidad sutilmente y son muy interesantes de cara a desarrollos futuros.

Otro aspecto por destacar es el uso de metodologías ágiles, las cuales no tuve oportunidad de estudiar ni emplear durante la carrera y me ha sido posible aplicarlas a este proyecto, considerando que proporcionan una forma de abordar un proyecto iterativamente, de forma que es fácil introducir cambios y hacer revisiones sobre los objetivos y metodologías que se están empleando antes de que suponga un gran problema incorporar estos cambios como podría suceder siguiendo una metodología tradicional. Si bien es cierto que es una metodología más enfocada a grupos de pocos participantes en lugar de este caso en el que los participantes han sido el director de este trabajo y yo mismo.

En conjunto he de decir que ha sido una experiencia muy enriquecedora tanto por los conocimientos adquiridos, la dinámica de trabajo seguida y por supuesto por trabajar con los profesionales que me han permitido que este trabajo pudiera llevarse a cabo.

8.4. LÍNEAS FUTURAS

Como trabajo futuro sería interesante integral el módulo de cálculo de probabilidades con la información obtenida en tiempo de ejecución y no únicamente con la información ejecuciones pasadas, lo que proporcionaría más flexibilidad al selector.

A su vez, la incorporación de nuevas métricas para realizar la selección, como por ejemplo el consumo de energía, la cantidad de recursos utilizados… que complementen al tiempo de ejecución, incrementarían la funcionalidad de la herramienta permitiendo extender su uso a áreas en las que estos factores cobran más relevancia que el tiempo de ejecución.

Otro desarrollo interesante podría consistir en desarrollar una herramienta de monitorización para poder tener una forma intuitiva de revisar el histórico de selecciones y poder hacer un análisis pormenorizado de la relación entre tamaño del problema y el dispositivo utilizado para su ejecución.

Por último, considero que sería fundamental tener un instalador de forma que se instalaran todas las librerías y dependencias necesarias para utilizar el framework de OmpSs y el planificador presentado en este proyecto. Es un aspecto que puede quedar relegado a un segundo plano, pero si se pretende que una herramienta tenga un uso masificado, se convierte en una necesidad fundamental que su instalación pase a ser algo trivial que no lleve al usuario a dedicar una gran cantidad de tiempo hasta tener la herramienta operativa.

9. ANEXOS

A. INSTALACIÓN

En este apéndice se va a explicar cómo instalar el framework OmpSs [32] y cómo añadir el selector de implementaciones basado en probabilidades en un entorno Linux.

OMPSS

1. Descarga de OmpSs

La página oficial de OmpSs contiene los enlaces para la descarga de todo su software. En el momento de la redacción de este documento se encuentran disponibles dos versiones de OmpSs.

Para este proyecto se utilizó la primera versión de OmpSs ya que era la última versión disponible en ese momento.

El enlace para descargar el software es el siguiente:

https://pm.bsc.es/ompss-downloads

2. Creación de carpeta de instalación

Para la instalación del framework de OmpSs se recomienda crear una carpeta donde se introduzca todos los componentes del framework, de esta forma será más sencillo configurar los programas que componen este framework.

En este caso se va a crear una carpeta llamada "ompss" dentro de la carpeta "home" del sistema. A continuación, se va a exportar la dirección de esta carpeta como variable de entorno, en este caso "TARGET", para facilitarla configuración de los componentes.

El comando para ello es el siguiente:

```
$ export TARGET=$HOME/ompss
```

3. Instalar Extrae

Extrae es parte del software de OmpSs y lo conforman una serie de herramientas para la instrumentalización del código.

Extrae no se incluye en la descarga de OmpSs y hay que acceder a la página de Barcelona Supercomputing Center para descargarlo:

https://tools.bsc.es/downloads

Se recomienda descomprimir el contenido dentro de una carpeta diferente a la de ompss. Una vez hecho, se ha de ejecutar el script de configuración pasando la variable de entorno TARGET para que obtenga la información de OmpSs.

Una vez terminado el script de configuración se ha de ejecutar el comando "make" y posteriormente "make install" para realizar la instalación.

```
$ ./configure --prefix=$TARGET
$ make
$ make install
```

4. Instalar Nanos++

El siguiente paso es instalar el runtime Nanos++.

Dentro de los archivos descargados de OmpSs se encuentra los archivos de Nanos++.

Se recomienda descomprimir su contenido en una carpeta diferente y ejecutar el script de configuración pasando la variable de entorno TARGET que hemos definido anteriormente.

Una vez que termina el script de configuración se debe ejecutar el comando "make" y "make install" para completar la instalación.

```
$ ./configure --prefix=$TARGET configure-flags
$ make
$ make install
```

5. Instalar Mercurium

A continuación, se va a instalar el compilador source-to-source Mercurium.

Este compilador no se encuentra en los archivos descargados de OmpSs y se debe descargas desde el repositorio de GitHub oficial.

Una vez descargado, se debe exportar la carpeta de Mercurium como variable de entorno para poder acceder a su contenido desde la terminal.

Posteriormente se ejecuta el script de configuración.

Ejecutar los comandos "make" y "make install" para finalizar la instalación.

Por último, añadir los binarios a la variable "PATH".

```
$ git clone https://github.com/bsc-pm/mcxx.git
$ export MERCURIUM=/path/to/install/mercurium
$ ./configure --prefix=$MERCURIUM
$ make
$ make install
$ export PATH=$MERCURIUM:$PATH
```

SELECTOR DE IMPLEMENTACIONES BASADO EN PROBABILIDADES

En esta sección se va a explicar los pasos para introducir el nuevo planificador en el framework de OmpSs. Para ello, se deben seguir primero los pasos descritos en el apartado anterior.

1. Incluir en Nanos++ el nuevo planificador

Para que Nanos++ pueda reconocer el planificador basado en probabilidades, es necesario que se copie el archivo que lo contiene a una carpeta específica, donde se encuentran el resto de los planificadores del framework.

"nanox-XXX\src\plugins\sched\probabilisticSelector.cpp"

2. Modificar Makefile

Para que Nanos++ incorpore el planificador y pueda ser utilizado por el runtime, es necesario modificar el archivo "**makefile.am**" de la carpeta "*nanox-XXX\src\plugins*" para incluir la información del nuevo planificador tal y como se muestra en la figura 18.

3. Ejecutar el comando "make" y "make install"

```
$ make
$ make install
```

Siguiendo estos pasos se instalaría un planificador adicional. Para instalar el selector basado en probabilidades sería necesario realizar los cambios en Mercurium y Nanos que se explican brevemente en la sección de Implementación para incluir el pragma adicional y que este sea reconocido por Nanos.

B. SUMMARY IN ENGLISH

INTRODUCTION

The goal of this project is to define a probabilistic based selector of implementations. The selector supports homogeneous and heterogeneous platforms.

With this in mind, it is important to define what is a heterogeneous platform and why is important to develop this kind of system.

A heterogeneous platform is a system that includes multiple types of processors. However, this doesn't imply that all the systems with multiple processors are heterogeneous, if all the processors share a type, then those systems are called homogeneous. The heterogeneity resides in the differences in the type of the processors.

To main characteristic that differentiates one processor for another is its main functionality. For example, there are processors that focus on matrix and vector multiplication, others are focus in the signal processing, etc.

Each kind of processor shows a group of advantages and disadvantages, that makes them adequate for specific set of tasks. With this premise, the heterogeneous systems want to unify in one system a variety of processors to use the most efficient one depending on the task. Is at this point when a implementation selector makes sense, it will be the one in charge to identify which processor is the best, according to a predetermined goal, to perform a task and make use of the advantages of the heterogeneous system.

An implementation selector is then a program or a fragment of a program that receives as input different versions of the same function or method and it uses a logic to determine which implementation will be used.

Depending on the development of the selector, it can contain one or more algorithms with different objectives, so the selection can change in base of those. Some examples could be to reduce the execution time, make a better use of the energy resources, avoid an idle processor, etc.

An important element of the implementation selector is the concept of task. In this context, a task is the unit over which the selector performs the selection. There are multiple definitions of tasks depending on the context, but in this one, a good definition is the one that can be found in the OpenMP documentation, that states that a task is the minimal unit of execution that can be managed independently by a selector. They are the chunks of code that can be identified throw a determined mechanism, for example through tags, and the compiler is able to detect them and execute them [4].

OBJECTIVES

The main objective of this project is the development of a tool that gives an answer to the challenge that is proposed by the heterogenous systems, taking into account the following points:

1. It shall be able to identify different implementations of the same function.
2. It shall be able to select dynamically the most efficient implementation according to a given problem.
3. It shall be able to store the information of the current and previous executions and use this information in the selection process.
4. It shall display an intuitive interface.

The first objective is linked to the essence of the heterogeneous systems, where a different selection of processors is unified to give multiple choices to solve a problem. Given that each processor is different of each other, a different implementation may be required, and because of this the selector must be able to identify them.

The second objective tries to make use of the different nature of the heterogeneous system processors to use them as an advantage to resolve a problem. Depending on the problem and the circumstances, a processor can find a solution in a more efficient way in terms of execution time.

The third objective aims to reduce the time involved in the selection of the most efficient implementation, to do so, its key to reuse the information of previous executions, so in the current selection the amount of executions required to find the optimal is reduced drastically.

The last objective is based in a main requirement for a tool that wants to be used by a large number of users. In this case, the target user is the developer that works with heterogeneous systems, which has a lot of options while developing a solution and could use a simple way to identify the best approach to solve a problem and spend more time in other issues of its projects.

To accomplish the previous objectives, this document contains the development of an implementation selector based on probabilities. The probabilistic model has been developed by the professor Javier Fernandez Muñoz.

DOCUMENT ORGANIZATION

Technological context

The first section of the document is an investigation exercise to determine the technological context in which this project is based. The existing tools to solve the problems associated with homogeneous and heterogeneous platforms will be discussed and their advantages and disadvantages will be discussed.

Agile methodologies

This section explains what an agile methodology is, describes in detail how the SCRUM methodology is defined, and which ones are its main benefits.

System Analysis

This section contains how the SCRUM methodology has been applied in this project and explains the resources used in the planning, development and the revision of the project.

System Design

This section contains the core of the project. The main of the elements that compose the implementation selector are described in detail and also the modifications that have been made in the OmpSs framework for its addition. It also contains the explanation of the selection algorithm.

Result Evaluation

To check the efficiency of the implementation selector, an evaluation of its performance is done using a multiplication of matrices exercise and a comparison with the selector "versioning" that is included in the OmpSs framework. This comparison wants to acknowledge if this proposed solution provides some advantages over the existing tools in this area.

Planification, budget and socioeconomic scope

In this section there is analysis of the cost associated to the project and the planification that has been made in its development. It also contains the implications of the project in the economic, social and environmental scopes.

Regulation framework

This section contains the different regulations that are related to this project.

Conclusions

To finish the document, a series of conclusions about the implementation selector will be discussed. It also contains a series of personal comments and the improvements that could be make in future works.

SYSTEM DESIGN

The implementations selector based on probabilities uses a variety of technologies that allows its working. In this section these elements will be discussed.

Programming language

The OmpSs framework gives support to three different programming languages, C, C++ and Fortran. Among these languages, the C++ language was selected for various reasons.

1. It's the main language used in the OmpSs framework, including the compiler Mercurium and the runtime Nanos++.
2. The different schedules included in the OmpSs framework, such as the "versioning" selector, is written in C++, so its helpful to have some reference code to learn how to develop and include a new one in the system.
3. The C++ programming language is one of the most extended programming languages and it has a large community of developers and users behind that eases the process of finding information and resources regarding the language.

Selection based on probabilities

The algorithm that the selector uses its based on associating a probability to each implementation of a function. This probability is determined based in the size of the problem that is being solved and the executions that had been carried out.

Considering that there can be multiple problem sizes, the probability model will define intervals of probabilities according to the problem size. Those intervals will have associated a probability that will be different depending on the implementation.

For example, a scenario with two implementations and two problem sizes will make the model to define two intervals, one for each problem size, and for each interval, two probabilities will be associated, because there are two implementations.

Once the problem size is obtained from a problem that is being executed, the algorithm has to choose one of the implementations, for that it will use the wheel strategy. This model will subdivide each interval in subintervals where the size of the subinterval is determined by the probability that it must be selected. Then a pseudo random number between zero and one is calculated and in base of it, the implementation will be selected.

Psize clause

As has been discussed in the previous section, the definition of the problem size is a critical aspect for the implementation selector. This value and the information from previous executions will determine which implementation is going to be selected.

The problem size is a parameter that must be introduced by the user. With this in mind is important to give the user a mechanism that allows him to provide this information in an intuitive way.

The solutions consists in the addition of a new clause to the OmpSs framework, that will be called “psize”, as a diminutive of problem size. This clause will determine which one is the parameter of the function that contains the problem size.

The next figure shows an example of it:

```
# pragma omp target device ( smp ) psize (2)
# pragma omp task
void func ( int **m, int problemSize );
```

Figura 14: Ejemplo cláusula psize

The “psize” clause is next to the target directive, that shows to the compiler in which device this function should be executed. The value of “psize” is “2”, so it means that the second parameter of the function called “func” is the one that contains the problem size.

The inclusion of this new clause requires the modification of the OmpSs framework, more precisely the Mercurium and Nanos++ components.

Probability Update Algorithm

The first important decision that has been made is which one is going to be the factor that determines which implementation is better over the others, and this one is the execution time, so the implementation that solves the problem in less time is going to be consider the best choice.

The next equation will determine which implementation shows the greater performance.

$$\text{Best(A; S)} = \forall i \in S : E(A) \leq E(i)$$

Equation 1: Selection of the implementation with greater performance

Given a set of implementations S, the implementation A has a performance greater than the set S if its execution time E(A) is lower than all the implementations in the set.

The probabilities calculation is based on the mean and the standard deviation over the execution time of each implementation, using it to define the trust intervals that are used to obtain their probability.

In the situation where two intervals are disjoint, the implementation with less execution time and therefore more performance, will be the one with a hundred percent of probabilities of being selected. However, in the scenario where the intervals are overlapped, the probabilities will be shared. The more executions that are mode, the more the probabilities of the intervals are reduced until the intervals are disjoint.

The next image shows an example for the calculus of the probability for three implementations.

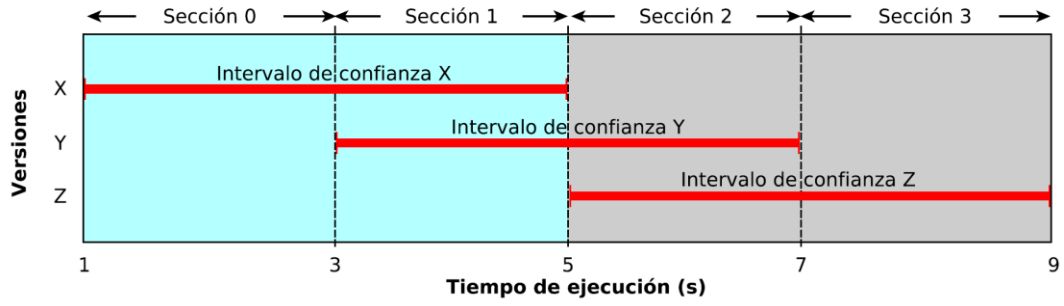


Figure 18: Probabilities calculation example

In this example there are three implementations, X, Y and Z. In the previous figure it is showed that each implementation has a trust interval that goes along the X axis and that represents the execution time.

The intervals are overlapped for some execution times. The different sections are defined by the values that determine the beginning and the end of the intervals, so there are four sections from zero to three.

The implementation X shows the trust interval that starts with the less execution time, so at this moment is possible to discard the sections that don't overlap this interval. This decision is made based on the fact that the next sections contains implementations that won't provide a performance better that the one displayed by the first implementation. So, in this case the section 0 and 1 are kept.

At this point, the relevant sections are well determined, so the next step is to use the law of total probability to obtain the marginal probabilities associated to each implementation. In this case, the implementation Z is not present in the selected sections, so the selector won't select this implementation in any case.

To obtain the marginal probabilities of each implementation in a particular section it's required to use again the law of total probabilities to compare each implementation with the others. There are three marginal probabilities that must be computed, one for the case where the execution time is contained in the selected section, one when its value is greater than the section and other when is lower.

The next equations show the calculation of the probabilities X, Y and Z. The variable I_n represents the section n, CI_j represents the trust interval for implementation j, and B_t and E_t represents the beginning and end of a section or trust interval respectively.

$$\mathbf{P}\big(Best(X,\{X,Y,Z\})\big) = \mathbf{P}\big(Best(X,\emptyset) \mid \mathbb{E}(X) \in I_0\big)\mathbf{P}\big(\mathbb{E}(X) \in I_0\big) +$$
$$\mathbf{P}\big(Best(X,\{Y\}) \mid \mathbb{E}(X) \in I_1, \mathbb{E}(Y) \in I_1\big)\mathbf{P}\big(\mathbb{E}(X) \in I_1\big)\mathbf{P}\big(\mathbb{E}(Y) \in I_1\big) +$$
$$\mathbf{P}\big(Best(X,\{Y\}) \mid \mathbb{E}(X) \in I_1, \mathbb{E}(Y) > I_1\big)\mathbf{P}\big(\mathbb{E}(X) \in I_1\big)\mathbf{P}\big(\mathbb{E}(Y) > I_1\big) =$$
$$1 \cdot \frac{E_{I_0} - B_{I_0}}{E_{CI_X} - B_{CI_X}} + \frac{1}{2} \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_X} - B_{CI_X}} \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_Y} - B_{CI_Y}} + 1 \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_X} - B_{CI_X}} \cdot \frac{E_{CI_Y} - E_{I_1}}{E_{CI_Y} - B_{CI_Y}} =$$
$$1 \cdot \frac{3-1}{5-1} + \frac{1}{2} \cdot \frac{5-3}{5-1} \cdot \frac{5-3}{7-3} + 1 \cdot \frac{5-3}{5-1} \cdot \frac{7-5}{7-3} = \frac{28}{32} = 0{,}875$$

Equation 2: Calculation of the probabilities of implementation X

$$\mathbf{P}\big(Best(Y,\{X,Y,Z\})\big) = \mathbf{P}\big(Best(Y,\{X\}) \mid \mathbb{E}(Y) \in I_1, \mathbb{E}(X) \in I_1\big)\mathbf{P}\big(\mathbb{E}(Y) \in I_1\big)\mathbf{P}\big(\mathbb{E}(X) \in I_1\big) = \frac{1}{2} \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_Y} - B_{CI_Y}} \cdot \frac{E_{I_1} - B_{I_1}}{E_{CI_X} - B_{CI_X}} = \frac{1}{2} \cdot \frac{5-3}{7-3} \cdot \frac{5-3}{5-1} = \frac{4}{32} = 0{,}125$$

Equation 3: Calculation of the probabilities of implementation Y

$$\mathbf{P}\big(Best(Z,\{X,Y,Z\})\big) = \mathbf{P}\big(Best(Y,\emptyset) \mid \mathbb{E}(Z) \in \emptyset\big) = 0$$

Equation 4: Calculation of the probabilities of implementation Z

CONCLUSIONS

Project conclusions

The implementation selector based on probabilities, that has been described in this document, supposes an improvement over the existing selectors for heterogeneous platforms because it delivers an intuitive and efficient way of defining the implementations and provides an efficient response, in terms of execution time, in an environment where the problems are mutable.

At this moment is important to evaluate if the objectives defined have been accomplish and to what extent.

1. It shall be able to identify different implementations of the same function.

This objective has been satisfied by using the pragmas as a mechanism to identify the base implementation and the different alternatives. In this way, the selector can recognize and use the alternatives to apply the algorithm of selection and choose the most appropriate option.

2. It shall be able to select dynamically the most efficient implementation according to a given problem.

This objective has been fulfilled using the algorithm based on probabilities. It evaluates at execution time the implementations and selects the most efficient in terms of execution time.

3. It shall be able to store the information of the current and previous executions and use this information in the selection process.

The storage of previous executions is made though a text file, that is access at the beginning and end of each execution, to retrieve and write the data accordingly.

4. It shall display an intuitive interface.

This point is considered to be achieved by using the “implements”, “target device” and “psize” clauses. They are an intuitive way of determine which implementation corresponds to which device and which is the problem size that should be consider to use the implementation.

FUTURE WORK

After the development of the project, there are some interesting improvements that could be done to increase the value and the utility of the solution.

First, the module that contains the calculations of the probabilities could be used with the information obtained in execution time and not only with the information of previous executions. This action would increase the flexibility of the selector.

Another interesting addition would be the inclusion of several options regarding the metrics involved in the selection process. Some of them could be the optimization of the resources consumption, the energy utilization by the system, etc. With these additional options the selector would be interesting to other sectors that don't have as primary target the reduction of the execution time of their applications.

As a complementary tool, a resources monitor would be interesting to have an efficient way of analyzing the data from the current executions and the historical data. This way an in deep study of the results could be made easily and could help in the decision making of topics such as selecting which kind of processor is more interesting depending which problem its being faced.

A very important feature that is missing, that is key in the wide use of a tool like this, is the addition of an automatic installer that contains all the dependencies, additional frameworks and modifications that require to make use of the implementation selector.

10. BIBLIOGRAFÍA

[1] S. Amar, «Heterogeneous Processing: a Strategy for Augmenting Moore's Law,» 2 Enero 2006. [En línea]. Available: https://www.linuxjournal.com/article/8368. [Último acceso: 22 Julio 2018].

[2] D. Riehle, «Framework Design: A Role Modeling Approach,» 2000. [En línea]. Available: https://riehle.org/computer-science/research/dissertation/diss-a4.pdf. [Último acceso: 12 Marzo 2019].

[3] M. Taylor, «What's Wrong with Libraries, and What to Do about It,» Abril 2010. [En línea]. Available: https://pragprog.com/magazines/2010-04/tangled-up-in-tools. [Último acceso: 12 Marzo 2019].

[4] «OpenMP 5.0 Complete Specifications,» Noviembre 2018. [En línea]. Available: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf. [Último acceso: 2 Febrero 2019].

[5] C. Mims, «Why CPUs Aren't Getting Any Faster,» 12 Octubre 2010. [En línea]. Available: https://www.technologyreview.com/s/421186/why-cpus-arent-getting-any-faster/. [Último acceso: 4 Febrero 2019].

[6] «Basic Linear Algebra Subprograms,» [En línea]. Available: http://www.netlib.org/blas/. [Último acceso: 22 Julio 2018].

[7] «clBLAS,» [En línea]. Available: https://github.com/clMathLibraries/clBLAS. [Último acceso: 22 Julio 2018].

[8] «cuBLAS,» [En línea]. Available: https://docs.nvidia.com/cuda/cublas/index.html#device-api. [Último acceso: 22 Julio 2018].

[9] W. J. Tan, W. T. Tang, R. Goh, S. Turner y W.-F. Wong, «A Code Generation Framework for Targeting Optimized Library Calls for Multiple Platforms,» *IEEE Transactions on Parallel and Distributed Systems*, nº 26(7), pp. 1789-1799, 2015.

[10] «The OmpSs Programming Model,» [En línea]. Available: https://pm.bsc.es/ompss. [Último acceso: 25 Julio 2018].

[11] J. Planas, R. M. Badia, E. Ayguade y J. Labarta, «Self-Adaptive OmpSs Tasks in Heterogeneous Environments,» *IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS*, pp. 138-149, 2013.

[12] «SkePu: Autotunable Multi-Backend Skeleton Programming Framework for Multicore CPU and Multi-GPU Systems,» [En línea]. Available: https://www.ida.liu.se/labs/pelab/skepu/skepu1/. [Último acceso: 25 Julio 2018].

[13] «Mercurium,» [En línea]. Available: https://github.com/bsc-pm/mcxx. [Último acceso: 5 Agosto 2018].

[14] «Nanos++ Runtime Library,» [En línea]. Available: https://github.com/bsc-pm/nanox. [Último acceso: 5 Agosto 2018].

[15] D. W. Anand Lal Shimpi, «anandtech,» 8 Noviembre 2006. [En línea]. Available: https://www.anandtech.com/show/2116/8. [Último acceso: 20 Marzo 2019].

[16] «CUDA Language Solutions,» [En línea]. Available: https://developer.nvidia.com/language-solutions. [Último acceso: 20 Marzo 2019].

[17] Khronos, «The OpenCL Specification v2.2-10,» 5 Febrero 2019. [En línea]. Available: https://www.khronos.org/registry/OpenCL/specs/2.2/html/OpenCL_API.html#_the_opencl_architecture. [Último acceso: 20 Marzo 2019].

[18] L. M. Sanchez, D. D. R. Astorga, M. F. Dolz y J. Fernandez, «CID: A Compile-Time Implementation Decider for Heterogeneous Platforms Based on C Attributes,» de *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress*, Toulouse, France, 2016.

[19] «REPARA: Reengineering and Enabling Performance and poweR of Applications,» [En línea]. Available: http://repara-project.eu/. [Último acceso: 9 Febrero 2019].

[20] U. Dastgeer, L. Li y C. Kessler, «"Adaptive implementation selectionin the SkePU skeleton programming library,» Agosto 2013. [En línea]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.698.6563&rep=rep1&type=pdf. [Último acceso: 7 Febrero 2019].

[21] «Agile Manifesto,» [En línea]. Available: http://agilemanifesto.org/. [Último acceso: 9 Febrero 2018].

[22] M. C. Layton, «10 key benefits of Scrum,» [En línea]. Available: http://www.dummies.com/careers/project-management/10-key-benefits-of-scrum/. [Último acceso: 14 Marzo 2018].

[23] «History of C++,» [En línea]. Available: http://www.cplusplus.com/info/history/. [Último acceso: 8 Agosto 2018].

[24] «Standard C++ Foundation,» [En línea]. Available: https://isocpp.org/about. [Último acceso: 8 Agosto 2018].

[25] «Intel Math Kernel Library,» [En línea]. Available: https://software.intel.com/en-us/mkl. [Último acceso: 26 Agosto 2018].

[26] C. P. d. I. e. I. d. l. C. d. Madrid, «Código Deontológico,» 20 Junio 2018. [En línea]. Available: http://www.cpiicm.es/wp-content/uploads/sites/3/2015/06/Codigo-Deontologico-CPIICM.pdf. [Último acceso: 9 Junio 2019].

[27] C. Commons, «Creative Commons,» [En línea]. Available: https://creativecommons.org/licenses/by-nc-sa/3.0/es/. [Último acceso: 9 Junio 2019].

[28] J. Fernández, A. Sánchez, D. d. R. Astorga, M. F. Dolz y J. D. García, «Probabilistic-Based Selection of Alternate Implementations for Heterogeneous Platforms,» *International Conference on Algorithms and Architectures for Parallel Processing*, nº 10393, 2017.

[29] «ICA3PP,» [En línea]. Available: http://nsclab.org/ica3pp2018/. [Último acceso: 23 Marzo 2019].

[30] J. Fernández, A. Sánchez, D. d. R. Astorga, M. F. Dolz y J. D. García, «Selección de implementaciones en plataformas heterogéneas basada en probabilidades,» de *Jornadas Sarteco*, Málaga, 2017.

[31] «Jornadas Sarteco,» [En línea]. Available: http://www.jornadassarteco.org/. [Último acceso: 23 Marzo 2019].

[32] «Installation of OmpSs,» [En línea]. Available: https://pm.bsc.es/ftp/ompss/doc/user-guide/installation.html. [Último acceso: 16 Marzo 2019].

[33] K. Rupp, «42 Years of Microprocessor Trend Data,» 15 Febrero 2018. [En línea]. Available: https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/. [Último acceso: 4 Febrero 2019].

[34] «OpenMP 4.0.0 Examples,» Noviembre 2013. [En línea]. Available: https://www.openmp.org/wp-content/uploads/OpenMP4.0.0.Examples.pdf. [Último acceso: 25 Julio 2018].

[35] R. M. Badia y X. Martorell, «Programming with OmpSs,» Octubre 2014. [En línea]. Available: https://www.bsc.es/sites/default/files/public/mare_nostrum/hpc-events/patc_ompss_2014.pdf. [Último acceso: 29 Julio 2018].

[36] «SkePU: Skeletons,» [En línea]. Available: https://www.ida.liu.se/labs/pelab/skepu/#map. [Último acceso: 7 Febrero 2019].

[37] J. Job, «Scrum Diagram,» 7 Diciembre 2017. [En línea]. Available: https://jordanjob.me/2015/12/07/scrum-diagram/. [Último acceso: 3 Marzo 2018].

[38] «Roulette wheel selection,» [En línea]. Available: http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php. [Último acceso: 15 Agosto 2018].

[39] R. M. Badia y X. Martorell, «Tutorial OmpSs: Development methodology and infrastructure,» 2013 Julio 12. [En línea]. Available: https://www.bsc.es/sites/default/files/public/computer_science/extreme_computing/tutorial_patc_oct_2013_dev.pdf. [Último acceso: 1 Abril 2019].

[40] «Statista,» 2019. [En línea]. Available: https://www.statista.com/statistics/748551/worldwide-households-with-computer/. [Último acceso: 9 Junio 2019].

[41] «GSMA,» 9 Junio 2019. [En línea]. Available: https://www.gsmaintelligence.com/. [Último acceso: 9 Junio 2019].