Escalabilidad de aplicaciones de simulación oceánica

Carmona Calpe, Carles

Resum– Este proyecto analiza y propone optimizaciones para el framework NEMO, de uso generalizado en la investigación del clima y de los océanos en los centros de investigación europeos. Se han añadido optimizaciones en el código actual usando diferentes tecnologías de paralelismo, permitiendo obtener las conclusiones desarrolladas en este documento, y justificando el uso de estas.

Paraules clau– Simulación oceánica, NEMO, Acelerador de cómputo, Computación Paralela, CUDA.

Abstract– This project analyzes and proposes optimizations for the NEMO framework of generalized use in the climate investigation for oceans in european investigation centers. The optimizations have been added in the code using different parallel technologies, obtaining the developed conclusions in this document and justifying their use.

Keywords– Ocean simulation, NEMO, Computing Accelerator, Parallel Computing, CUDA.

1 Introducción

NEMO (Nucleus for European Modelling of the Ocean) es un framework usado para el estudio oceanográfico y de clima, de uso común en la comunidad científica de estos ámbitos. En la actualidad, más de 240 proyectos que involucran a 27 países diferentes utilizan NEMO.

El framework está compuesto por engines, que proporcionan soluciones numéricas a ecuaciones y físicas relativas al océano, las banquisas (hielo marino) o el estudio de la biogeoquímica.

En el modelo actual, se paraleliza el cómputo de manera que cada procesador realiza el cómputo relativo a un subdominio. Para la comunicación entre estos subdominios se usa la librería OpenMPI.

En la actualidad se esta investigando desde diferentes ámbitos europeos [1][2][3] la evolución de NEMO para la computación Exascale, donde los equipos permiten realizar 10^{18} cálculos por segundo.Esta evolución se debe a la necesidad de aplicar nuevos modelos matemáticos mas complejos y que permitan mejor detalle en la simulación.

E-mail de contacto: carlescarmonacalpe@gmail.com
Mención realizada: Ingeniería de Computadores
Proyecto tutorizado por: Ana Cortés y Juan Carlos Moure (CAOS)

En la Figura 1 se puede observar el resultado de una simulación en NEMO con los mismos engines, en el gráfico de la izquierda con resolución 1/4º, es decir, 0,25 grados entre cada subdominio y en el gráfico de la derecha con resolución 1º.

Claramente el gráfico de la izquierda muestra mucho mas detalle y justifica la necesidad de optimizar el framework.

En la Figura 2 se observa como de relevante es la resolución del dominio, ya que el aumento de la resolución provoca un aumento de recursos físicos y computacionales.

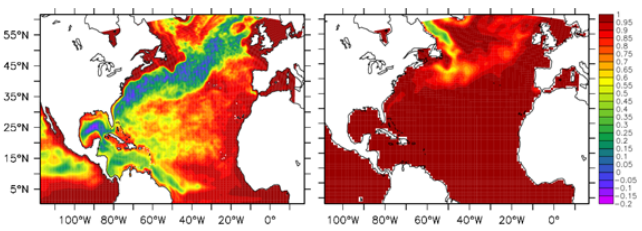

Figura 1: Correlación de la altura del mar en 5 días con dos resoluciones del dominio

Ademas, como hecho mas destacable, aún no hay conclusiones/soluciones globales para la falta de escalabilidad del framework, que provoca que los tiempos de ejecución para resoluciones muy altas sean ilógicos, en otras palabras, en muchos estudios se confirma que es necesario un análisis

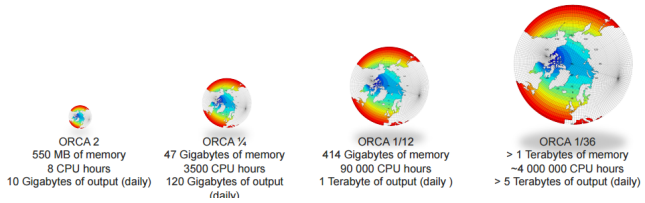

Figura 2: Evolución de los requerimientos con diferentes resoluciones del dominio.[4]

profundo y actuación en los siguientes ámbitos:

- Entrada/Salida.
- Mejora en la comunicación, tanto en la implementación como en el diseño.
- Programación híbrida.
- Aceleradores de cómputo.

En conclusión, la falta de escalabilidad y las necesidades del problema permiten afirmar que hace falta un replanteamiento de la arquitectura y el tipo de paralelismo que debe explotar el framework.

2 Estado del arte

Para configurar los datos de entrada y los modelos matemáticos usados en la ejecución del framework existen diferentes configuraciones, ejemplos de estas configuraciones son ORCA,ORCA2,ORCA2_LIM_PISCES o GYRE que permiten ajustar la resolución del dominio.

Respecto al dominio de datos en la Figura 3 se puede observar como este se descompone en forma de malla plana, en otras palabras, un plano donde los ejes representan longitud y latitud, y donde cada subdominio contiene datos relativos a la representación del modelo. Por ejemplo, la temperatura o concentración de un producto químico en un cierto nivel de profundidad.

Un aspecto relevante en el framework es la resolución de la descomposición que antes mencionaba, es decir, cada cuantos grados empieza un nuevo subdominio, ya que la resolución sera directamente proporcional al paralelismo generado por los datos.

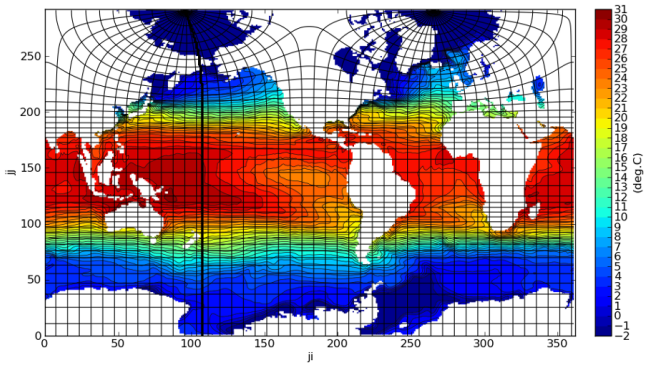

Figura 3: Descomposición del dominio con la configuración ORCA2

Durante todo el proyecto se ha usado la configuración GYRE debido a su semejanza con ORCA, de uso generalizado, aunque con la diferencia de que GYRE usa un input sintético, es decir, generado por el propio framework en cada inicio de simulación.

En la actualidad se investiga como llevar la ejecución del framework a resoluciones de 1/16º o 1/32º de manera óptima y maximizando el uso de los recursos hardware.

Es por eso que las nuevas tecnologías aportan una nueva visión de diseño para el núcleo del framework con diferentes modelos de paralelismo y permiten investigar nuevos caminos para problemáticas del framework como son la comunicación, la entrada/salida y la falta de escalabilidad.

3 Esquema del artículo

El artículo se organiza en diferentes secciones independientes, permitiendo una lectura en diagonal, de forma que el lector pueda obtener las ideas principales del proyecto de manera fácil y rápida.

La primeras secciones hacen referencia a los objetivos y la metodología seguidas en el proyecto, a continuación se desarrolla la parte más técnica del proyecto con diferentes secciones directamente relacionadas con los objetivos. En cada una de estas secciones se puede apreciar el siguiente esquema: resumen, problemática, desarrollo y conclusiones.

Finalmente, se exponen las conclusiones y líneas de trabajo futuras para el proyecto y además un apéndice con las diferentes plataformas de ejecución del proyecto.

4 Objetivos

El objetivo principal de este proyecto es analizar y optimizar las problemáticas mas relevantes del framework NEMO, concretamente, la falta de escalabilidad al aumentar la resolución del framework y el excesivo incremento de comunicación al paralelizar la ejecución del framework. A continuación se detallan los objetivos de forma específica:

- Implementar una propuesta de comunicación asíncrona que substituya el protocolo actual [5] y ejecutar el framework NEMO modificado en el supercomputador Marenostrum, analizando los resultados de la ejecución con distintas configuraciones.
- Realizar estudios de ingeniería de rendimiento en el framework NEMO, y obtener una lista de funciones del núcleo del código determinantes en el tiempo de ejecución.
- Proponer optimizaciones con diferentes tecnologías actuales para mejorar el tiempo de ejecución del framework mediante paralelismo.
- Analizar el uso de aceleradores de cómputo para la ejecución del framework NEMO.

5 Metodología

Para el proyecto se usará la metodología de programación iterativa e incremental [6][7], que se caracteriza por un conjunto de tareas agrupadas en etapas iterativas.

He escogido la metodología de programación iterativa e incremental por las siguientes ventajas:

- Entregable funcional en cada final de etapa.
- Posibilidad de ajustar el entregable según la necesidad.
- Alcance acotado y con viabilidad.
- Visualización del proyecto en un corto periodo de tiempo

6 Mejora del protocolo de comunicación

En esta sección se expondrá la problemática actual en el protocolo de comunicación del framework NEMO, una propuesta de solución, el entorno experimental y por último los resultado y las conclusiones obtenidas en este proceso.

6.1 Problemática en la comunicación actual

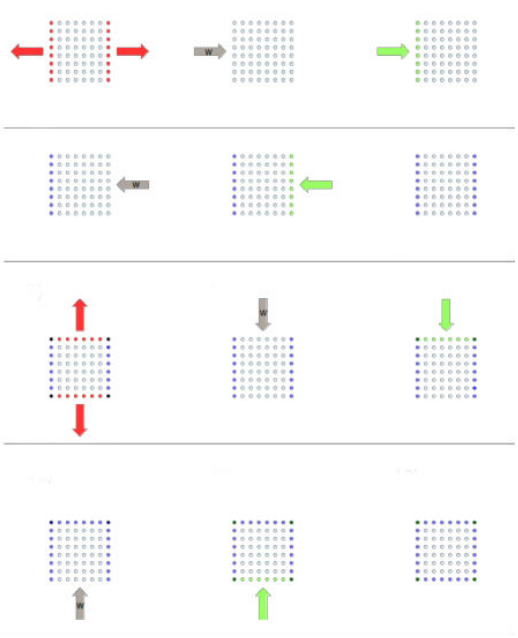

Figura 4: Comunicación en la versión actual de NEMO

En la Figura 4 se representa el proceso de comunicación original del modelo, donde las flechas en rojo simbolizan el envió de datos, las flechas en gris la espera de datos y las flechas en verde la recepción de datos.

El protocolo envía un conjunto de datos a las fronteras este-oeste,después en varias operaciones recibe un conjunto de datos de de sus vecinos este-oeste y luego realiza el mismo proceso para las fronteras norte-sur.

Las operaciones se realizan de forma secuencial y en orden, los símbolos en gris implican tiempo de espera debido a la rutina de recepción bloqueante (Mpi_Recv), que no permite la ejecución de la siguiente instrucción hasta recibir el siguiente mensaje.

Esto permite deducir a priori que el framework no escala como debería por una implementación no muy adecuada, y efectivamente en la Figura 5 se aprecia como aumentar el numero de procesadores no reduce proporcionalmente el tiempo de ejecución.

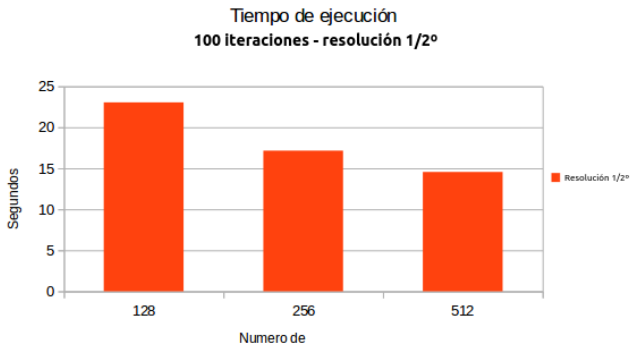

Figura 5: Tiempo de ejecución del framework en Marenostrum

6.2 Propuesta de comunicación

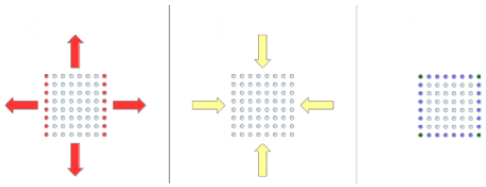

Figura 6: Propuesta de comunicación

La Figura 6 representa el mismo protocolo de comunicación original pero con instrucciones no bloqueantes.

Las flechas en rojo representan el envió de datos y las amarillas las rutina de recepción no bloqueante que consiste en una rutina, que continua el proceso de la ejecución pero recibiendo el mensaje una vez se notifica su llegada, a diferencia del método bloqueante antes de usar los datos que recibimos hay que comprobar que han llegado mediante un flag, esto permite el solapamiento de otras operaciones que no dependan de los datos hasta la llegada del mensaje.

La implementación del protocolo representado en la Figura 6 se ha desarrollado con OpenMPI, modificando las rutinas originales de comunicación. De tal manera que todos los envíos se producen en una misma operación, y no agrupados en norte-sur y oeste-este, y el recibo de datos se produce de forma no bloqueante como antes he descrito.

6.3 Entorno experimental

Para familiarizarme con el entorno consulté bibliografía disponible sobre las herramientas más relevantes desarrolladas por el BSC [8][9], el gestor de colas[10] y además asistí a un workshop de programación en Marenostrum III[11].

Para hacer depuración he usado las herramientas Extrae y Paraver. Extrae permite extraer trazas de una ejecución para analizarla posteriormente: para lograrlo se precarga una librería en el sistema operativo que se interpone entre la aplicación y la librería OpenMPI,OpenMP o CUDA y registra todos los eventos. Paraver no es más que un visualizador que permite representar eventos relevantes de la ejecución del proceso.

Además he usado un parámetro (nn_timing) en la configuración del framework, concretamente en el fichero (namelist_cfg), que crea un fichero time.output al finalizar

la ejecución.Este fichero contiene para cada uno de los métodos su tiempo de ejecución.

Para afinar mas en el estudio de los tiempos de ejecución se ha modificado en los métodos la línea de código donde se iniciaba y paraba el contador, de tal manera que este no se vea influido por la inicialización de los datos de entrada sintéticos característicos de GYRE.

Una de las características relevantes del entorno es el uso del gestor de colas Slurm, esto permite configurar algunos parámetros en la ejecución paralelo.

En mi caso, ajuste un parámetro que permite que el numero de procesadores donde se ejecutaba el framework debía maximizar la ocupación de los nodos, de tal forma que si un nodo tiene 16 procesadores y se habían solicitado 32 procesadores, el framework se tenia que ejecutar en solo dos nodos.

Decidí usar esta configuración para intentar minimizar las comunicaciones a través de la red Infiniband.

6.4 Resultados y conclusiones

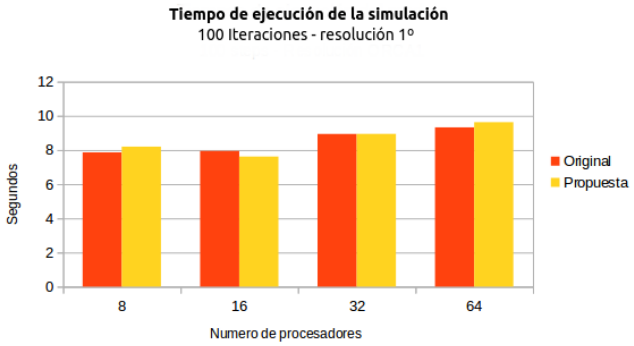

Figura 7: Resultado de la simulación con la resolución 1º

En la Figura 7 se aprecia el tiempo de ejecución de la simulación con una resolución de 1 grado, de color naranja con el protocolo de comunicación original y de color amarillo con el protocolo implementado.

En esa misma figura se observa como el framework no escala adecuadamente en ninguna de los dos versiones, y además se observa que los resultados no siempre favorecen una de las implementaciones, por ejemplo, con 16 procesadores donde el modelo propuesto tiende a mejorar la implementación original.

Esto es debido a varios problemas :

El desbalanceo de carga provocado por la división del dominio, ya que aunque en la introducción se ha descrito como se descompone el dominio, dependiendo del modelo que ejecuta el framework puede haber diferencias en el tiempo de cómputo en cada subdominio, por ejemplo, en el caso de el modelo que calcula el deshielo, en sectores próximos a los polos deberán realizarse muchas mas operaciones que en sectores en los trópicos.

Este desbalanceo de cómputo provoca que la aplicación escale mejor o peor según la configuración del numero de procesadores a usar. En la Figura 8 se aprecia el tiempo de ejecución de la simulación, de color naranja con el protocolo de comunicación original y de color amarillo con el protocolo implementado y con una resolución de 1/2º.

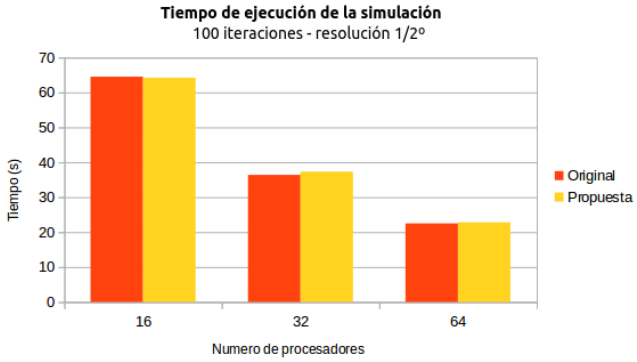

Figura 8: Resultado de la simulación con la resolución 1/2º

En este caso la escalabilidad es mucho mas adecuada debido a que la configuración se aproxima a un caso ideal por la resolución y numero de procesadores seleccionados.

Otro de los problemas en la comunicación, es compartido con la mayoría de aplicaciones HPC, donde el aumento de procesadores hace crecer de forma excesiva la comunicación entre nodos, ocupando por ejemplo mas del 40% del tiempo de ejecución total en una simulación con 128 procesadores como se puede apreciar en la Figura 9.

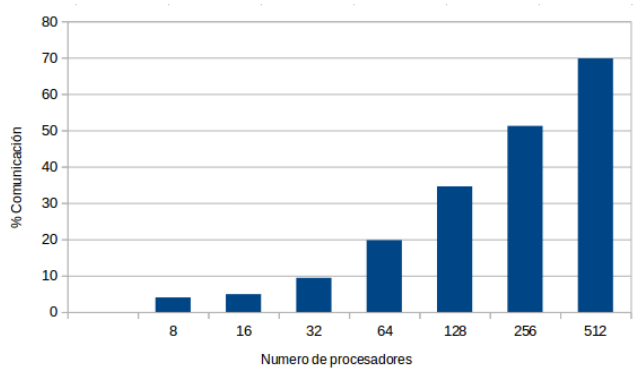

Figura 9: Porcentaje de comunicación con diferentes configuraciones de procesadores

En conclusión, la escalabilidad del framework NEMO no mejora con una comunicación asíncrona como se ha demostrado en la Figura 7,8 porque no se aprovechan los tiempos de espera para solapar operaciones de cómputo relacionadas con los modelos, es por eso que la mejora del protocolo de comunicación va ligado a tres aspectos:

- Reducir el numero de comunicaciones al mínimo.
- Solapar operaciones de cómputo con envíos que no tengan dependencias de datos.
- Unir datos para maximizar el volumen de datos a enviar.
- Balancear la carga ejecutando varios modelos de forma simultanea.

7 Ingeniería de rendimiento

En esta sección se presentan propuestas para mejorar el rendimiento de diferentes funciones del núcleo del código involucradas en un paso de simulación.

Primero se muestran los resultados del análisis de un paso de simulación utilizando el profiler del framework NEMO.

Después se muestran los resultados de una implementación con OpenMP, y finalmente se proponen mas optimizaciones para la versión serie.

7.1 Análisis de un paso de simulación

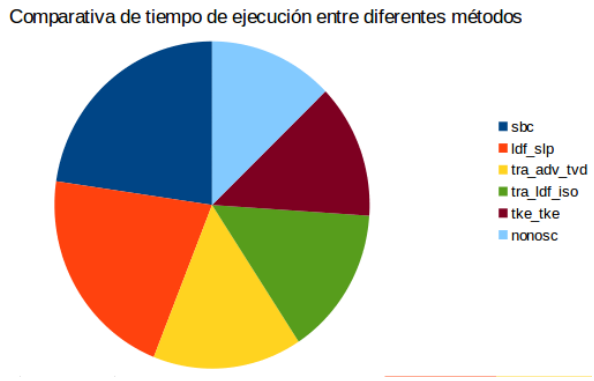

Figura 10: Tiempos de ejecución por método

En la Figura 10 se representa el tiempo de ejecución de las funciones que mas tiempo ocupan en un paso de simulación, solo aparecen seis funciones debido a que el resto de funciones ocupan menos de 4% del tiempo de ejecución total.

Los cinco métodos mas destacados por tanto son :

- dyn_spg_ts = Computo relacionado con el gradiente de presión en la superficie de la malla.
- dia_wri = Escritura de ficheros de salida.
- sbc = División de la superficie de la malla.
- ldf_slp = Computo relacionado con el calculo de físicas del océano.
- nononsc = Aplica el algoritmo nonoscillatory.

A continuación vamos a proponer una serie de propuestas para optimizar el modelo con diferentes técnicas.

7.2 Paralelismo: OpenMP

OpenMP (Open Multi-Processing) es una especificación de un conjunto de directivas para el compilador,rutinas de librería y variables de entorno que permiten especificar paralelismo a alto nivel en lenguajes de programación como Fortran,C o C++.

OpenMP se basa en que el programador se de cuenta de en que regiones se puede dividir el trabajo,definiendo que variables pertenecen a cada uno de los núcleos y cuales de ellas son compartidas.

Para la división de tareas se usa el modelo fork-join,donde una tarea muy pesada se divide en diferentes ramas de ejecución para luego recolectar los resultados.

En la Figura 11 se puede observar un ejemplo de uso de las directivas OpenMP.

Esta especificación puede suponer una mejora en el modelo ya que podría acelerar la parte secuencial del código y

```
DO jk = 1, jpkm1                    ! ------------------------------------------------
      DO jj = 2, jpjm1
            !$OMP PARALLEL DO
            DO ji = fs_2, fs_jpim1   ! vector opt.
                  ua(ji,jj,jk) = ua(ji,jj,jk) - zu_frc(ji,jj) * umask(ji,jj,jk)
                  va(ji,jj,jk) = va(ji,jj,jk) - zv_frc(ji,jj) * vmask(ji,jj,jk)
            END DO
            !$OMP END DO
      END DO
END DO
```

Figura 11: Fragmento de código con directivas OpenMP

optimizar el uso del procesador. Usando una combinación híbrida entre OpenMPI y OpenMP se podría conseguir que el framework escalara mejor, aunque muchos de los problemas mencionados antes aún seguirían.

Este tipo de programación híbrida ya se ha usado con el modelo obteniendo resultados satisfactorios [**?**].

7.3 Resultados y conclusiones

Para justificar el uso de la especificación OpenMP he implementado una versión de una de las funciones que mas tiempo requerían, concretamente nonosc, y he aplicado técnicas de optimización serie y directivas OpenMP.

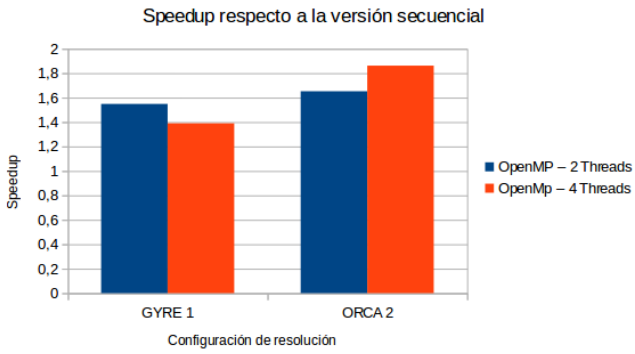

Figura 12: Speedup de la implementación OpenMp

Se aplicado un bucle externo al método para simular un numero de pasos de simulación determinado simulando el framework.

En la Figura 12 se muestra el speedup de la implementación en OpenMP. Se observa una mejora respecto a la versión serie pero el speedup no llega al ideal que en caso de 2 threads seria un speedup de 2x y en el caso de 4 threads 4x.

En la configuración GYRE con resolución 1º, con la versión de 2 threads se observa un speedup aceptable pero en el caso de los 4 threads escala muy mal.

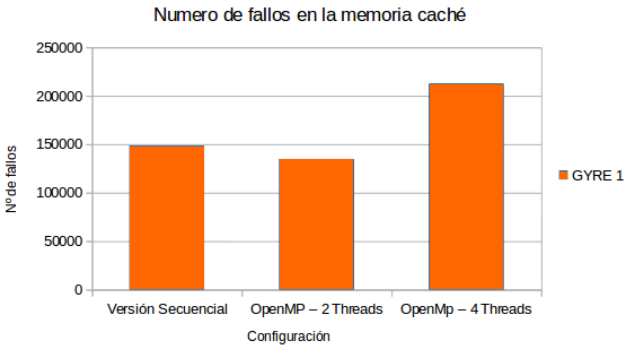

Figura 13: Fallos de caché de la implementación OpenMp

En la Figura 13 se observa la causa de que el speedup en la versión de 4 threads no mejore y es debido a un incremento del 43% en los fallos de acceso a la memoria caché, eso es debido a que la función usa muchas variables distintas en una misma operación de forma caché no se usa de la forma adecuada, ya que además compite con el otro thread para usar la memoria.

De todas maneras hay que tener en cuenta que esta implementación OpenMP obtendría resultados muy diferentes dependiendo de la resolución , ya que actualmente al usar OpenMPI se divide el dominio y cada procesador calcula solo su parte,de manera que por ejemplo en una ejecución del framework con 16 procesadores en ORCA1, esta implementación probablemente seria inadecuada, debido a que solo trataría 1/16 partes del volumen de datos.

7.4 Otras optimizaciones

Como hemos visto en la Figura 10, uno de los métodos que mas porcentaje de tiempo ocupa es el dyn_spg ts,este método realiza operaciones computacionales con los inputs del framework Nemo en cada paso de simulación.

El código tiene mas de una veintena de iteraciones, algunas de ellas con hasta tres niveles de profundidad,además también realiza diferentes operaciones fácilmente vectorizadas.

Es por eso que intenté usar las siguientes técnicas:

- Loop fusion

Esta técnica consiste en fusionar dos iteraciones adyacentes en un sola, de esta forma se reduce la sobrecarga de las iteraciones, es decir, las instrucciones extras para controlar las condiciones, y de esta forma poder mejorar sustancialmente el tiempo de ejecución, aunque en algunos casos puede desfavorecer la ejecución por culpa de no aprovechar correctamente los accesos a la memoria caché si los loops que se fusionan no usan variables en común.

En la Figura 14 se puede observar un ejemplo de la aplicación antes de fusionar las iteraciones.En la Figura 15 una vez aplicada la técnica.

```
DO jj = 1, jpjm1
        zhf(:,jj) = zhf(:,jj)*(1._wp- umask(:,jj,1) * umask(:,jj+1,1))
END DO

DO jk = 1, jpkm1
        DO jj = 1, jpjm1
                zhf(:,jj) = zhf(:,jj) + fse3f_n(:,jj,jk) * umask(:,jj,jk) * umask(:,jj+1,jk)
        END DO
END DO
```

Figura 14: Fragmento de código antes de aplicar la técnica loop fusion

```
DO jk = 1, jpkm1
        DO jj = 1, jpjm1
                zhf(:,jj) = zhf(:,jj)*(1._wp- umask(:,jj,1) * umask(:,jj+1,1))
                zhf(:,jj) = zhf(:,jj) + fse3f_n(:,jj,jk) * umask(:,jj,jk) * umask(:,jj+1,jk)
        END DO
END DO
```

Figura 15: Fragmento de después antes de aplicar la técnica loop fusion

En mis ensayos el resaltado no fue positivo, ya que apenas mejoraba un 1% el tiempo de ejecución.

- Loop Unrolling

Esta técnica consiste en intentar optimizar el tiempo de ejecución escribiendo implícitamente un numero de accesos mayor a un vector de datos dentro de una iteración, minimizando la predicción de saltos y también mejorando el acceso a caché, debido a que por ejemplo en una iteración si hay cuatro accesos a memoria y uno de ellos no se encuentra en la memoria caché los otros tres si.

En mis ensayos el resaltado no fue positivo, ya que apenas mejoraba un 3% el tiempo de ejecución.

8 Aceleradores de cómputo

En esta sección se presenta una implementación de una función del núcleo del framework NEMO usando aceleradores de cómputo.

Se justifica el uso de aceleradores de cómputo mediante los resultados de la implementación, y se simulan los beneficios en el tiempo de ejecución en una supuesta ejecución híbrida con OpenMPI.

8.1 Definición

Un acelerador de cómputo es un coprocesador diseñado para el procesamiento de gráficos o operaciones de coma flotante.La arquitectura es muy diferente que la de un procesador ya que los aceleradores gráficos tienen una visión de paralelismo masivo,donde pequeños threads se ejecutan simultáneamente,esto quiere decir que para alcanzar una eficiencia que justifique el uso de aceleradores de cómputo, hace falta una ejecución con miles de threads.

CUDA (Compute Unified Device Architecture) es un conjunto de herramientas que permiten al programador usar una variación de C para codificar algoritmos en aceleradores gráficos de la compañía NVIDIA.

OpenACC es un standard de programación basado en directivas para la programación paralela diseñado por Cray, CAPS, Nvidia y PGI. Es muy parecido a OpenMp, ya que el desarrollador tiene que identificar y definir las regiones paralelas,independientemente de la arquitectura donde se ejecute la aplicación.

Uno de los objetivos del proyecto era el análisis de la posibilidad de portar diferentes fragmentos o la totalidad del código a aceleradores gráficos, varios estudios [12][13][14] han desarrollado propuestas, aunque pocos parecen justificar una mejora en el tiempo de ejecución en las implementaciones con aceleradores de cómputo.

Los problemas principales son la excesiva cantidad de comunicaciones,el uso ineficiente de los aceleradores y la dificultad de depurar la ejecución.

8.2 Paralelismo : Nonosc

El método Nonosc tiene como objetivo aplicar el algoritmo nonoscillatory, para ello se siguen los siguientes pasos:

- Buscar el valor máximo y mínimo en dos vectores aplicando una mascara a estos, es decir, por cada posición determinar cual de los dos vectores tiene un valor máximo o mínimo
- En un volumen de datos de 3 dimensiones buscar para cada elemento el máximo y mínimo en sus vecinos y realizar operaciones vectoriales.
- Mas operaciones con vectores en 3 dimensiones.

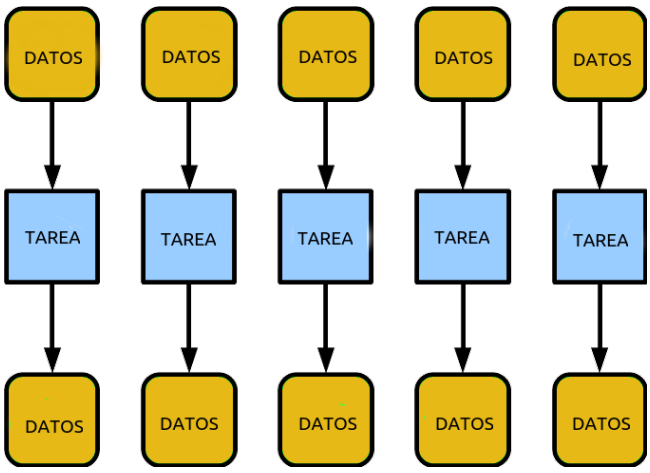

Figura 16: Patrón de diseño paralelo MAP

He escogido esta función por que: es fácil de comprender su funcionalidad sin conocer detalles del ámbito científico del framework, satisface la necesidad del uso masivo de threads y además se relaciona perfectamente con el patrón de diseño paralelo MAP.

En la Figura 16 se representa el patrón de diseño, donde un conjunto de datos son tratados de forma independiente y se obtiene un resultado para cada uno de ellos.

8.3 Resultados

Para justificar el uso de aceleradores he implementado una versión de esta función en C de la misma forma que en el caso de OpenMP, y he realizado una aproximación del tiempo de ejecución para una versión usando aceleradores de cómputo.

En la aproximación se generan datos aleatorios y se realiza una o varias ejecuciones del método nonosc.

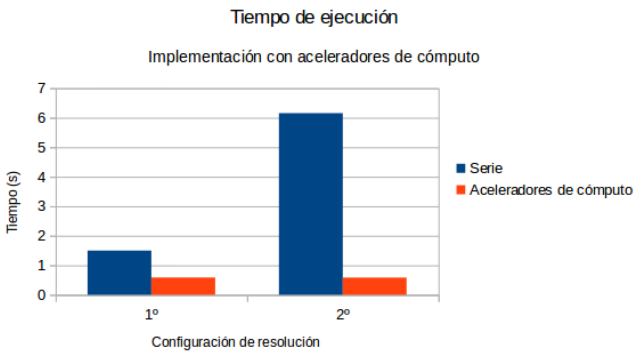

Figura 17: Tiempo de ejecución implementación secuencial y con aceleradores de computo con distintas configuración de resolución

En la Figura 17 se observa que el tiempo de ejecución en la versión con aceleradores de computo es más rápida en todas las resoluciones donde se han realizado pruebas.

En el caso de la configuración de la malla con resolución de 2^o se obtiene un speedup de 2,5x y en el caso de la configuración de la malla con la resolución de 1^o un speedup de mas de 10x.

En la Figura 17 también se observa la idea de paralelismo de la arquitectura de los aceleradores de cómputo ya que aunque se aumente considerablemente el volumen de datos el tiempo de ejecución no aumenta, esto es debido a que los aceleradores de cómputo se fundamentan en el paradigma de paralelismo masivo donde muchos threads ejecutan instrucciones simultáneamente en muchos datos permitiendo como en este caso tiempos de ejecución muy parecidos comparando dos volúmenes de datos.

8.4 Conclusiones

Con esta aproximación se ha demostrado que la implementación de la función usando aceleradores de cómputo esta justificada en el framework NEMO, pero hay que recordar que el volumen de datos dependerá directamente de la división del dominio necesaria para la paralelización con OpenMPI.

Para intentar valorar este problema he ajustado una configuración de malla con resolución de $1/4^o$ y he realizado una serie de pruebas con el volumen de datos que ocuparía un nodo, por ejemplo, en una configuración de ejecución paralela con 16 procesadores cada subdominio tendría un volumen de datos de 45x32x10.

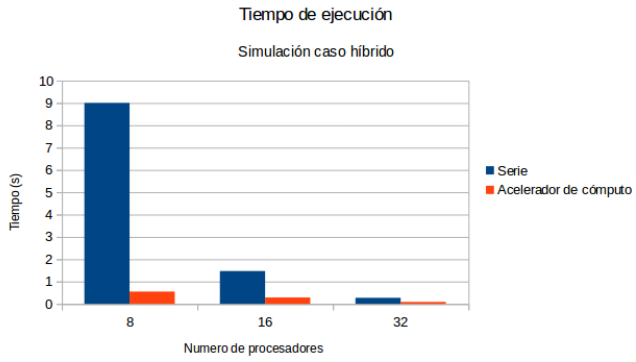

Figura 18: Tiempo de ejecución implementación secuencial y con aceleradores de cómputo simulando un caso híbrido con una resolución de $1/4^o$

En la Figura 18 se observa como todo y reduciendo el volumen de datos la implementación con aceleradores de cómputo obtiene una mejora en el tiempo de ejecución relevante, y justifica el uso de forma híbrida de la paralelización con OpenMPI y también con aceleradores de cómputo, que en conjunto mejorarían el tiempo de ejecución del framework NEMO.

9 Conclusiones Finales

En este proyecto he analizado el framework NEMO, una aplicación HPC que comparte una tendencia actual muy relevante, la optimización de código paralelo para adaptarlo a nuevas arquitecturas como los aceleradores de cómputo o los HPC Exascale.

Durante todo el proyecto he realizado análisis y propuestas para algunas de las problemáticas actuales del framework NEMO, obteniendo algunos resultados favorables y otros no favorables.

Se ha demostrado la necesidad de solapar cómputo durante la comunicación de los diferentes procesadores para aprovechar los tiempos de comunicación, y también la necesidad de reducir la cantidad de comunicación debido al

alto porcentaje de tiempo usado para esta durante la ejecución del framework.

Respecto a la versión implementada con las directivas OpenMP se ha demostrado que no aprovecha el paralelismo, al menos en la función que se ha implementado, y que de esta manera su uso para paralelizar cómputo en la parte serie del código no es recomendable.

Claramente, la versión de cómputo con aceleradores muestra los mejores resultados, y además justifica la necesidad de volver a implementar otras funciones del núcleo del código para obtener resultados similares.

Pero, en esencia, en el proyecto se han analizado las diferentes problemáticas que actualmente limitan el framework NEMO. Además se han explorado diferentes caminos para la paralelización del código, que me han permitido enfrentarme a arquitecturas muy diversas con puntos de vista distintos para la paralelización de la aplicación.

10 LÍNEAS FUTURAS

Hay tres líneas de investigación muy claras después de la realización del proyecto:

- Comparación entre el uso de OpenMP y OpenMPI para realizar la paralelización de operaciones vectoriales.
- Implementación de otras funciones del núcleo del framework NEMO en aceleradores de cómputo para agilizar la parte secuencial del framework e ir hacia un modelo híbrido con OpenMPI+Aceleradores de cómputo.
- Volver a implementar el protocolo de comunicación, intentando minimizar las comunicaciones para reducir el tiempo total de comunicación en la ejecución del framework y maximizar la cantidad de datos a enviar, es decir, empaquetar todos los datos siempre que sea posible para evitar repetidas comunicaciones con datos pequeños que se podrían agrupar y que hacen un uso inadecuado del hardware.

AGRADECIMIENTO

El autor quiere aprovechar este espacio para agradecer a mi familia el apoyo aportado durante la ejecución del proyecto, a el tutor Juan Carlos Moure por su competencia y sinceridad y a Ana Cortés por la oportunidad de realizar este proyecto.

REFERÈNCIES

[1] Reid, Fiona J.L., "The NEMO ocean modelling code: A case study." Proceedings of Cray User Group (CUG) 2010, 11 pages. http://www2.epcc.ed.ac.uk/ fiona/publications/Reid_paper.pdf.

[2] Epicoco, Italo, "NEMO-MED: Optimitzation and improvement of scalibility." Division ASC - Advanced Scientific Computing Division. Research Report, RP0096 (2010). http://www.cmcc.it/wp-content/uploads/2013/05/rp0096-sco-01-2011.pdf.

[3] Borovska,Plamenka and Ivanova,Desislava, "Scaling of software package NEMO for oceanology on the supercomputer JUQUEEN." Computer Systems Department at the Technical University of Sofia. (2013). http://goo.gl/B4eZAn.

[4] Vigilant, Franck, "Optimising applications and preparing exascale with bull technology." ENES Workshop, Hanmburg. (2014). https://goo.gl/fQ8Csw.

[5] Tintó Prims, Oriol, "Accelerating nemo: Towards exascale climate simulation," Master's thesis, Universitat Autònoma de Barcelona, 2014.

[6] Larman, C., Vodde, B, "Practices for scaling lean agile development: Large, multisite, and offshore product development with large-scale scrum."

[7] H.D. Mills, "Top-down programming in large systems."

[8] BSC, "Nemo - user guide manual for version 2.5.1." Barcelona Supercomputing Center 2014, 77 pages. http://goo.gl/Jci6yJ.

[9] BSC, "Paraver - detailed material." Barcelona Supercomputing Center 2014, 62 pages. http://goo.gl/52L7m1.

[10] BSC, "Marenostrum iii user's guide." Barcelona Supercomputing Center 2015. http://www.bsc.es/support/MareNostrum3-ug.pdf.

[11] BSC, "Patc course - material." Barcelona Supercomputing Center 2015. http://www.bsc.es/support/PATC-MareNostrum3/.

[12] Appleyard, Jeremy , "Accelerating nemo using openacc." NVIDIA 2014, 25 pages. http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/nvidia-ws-2014/04-appleyard-nemo.pdf?$_{blob=publicationFile}$.

[13] Porter, Andrew and Pickles, Stephen andAshworth, Mike , "Porting a fortran oceanographic code to gpus; the gnemo project." NVIDIA 2011, 32 pages. http://www.doc.ic.ac.uk/UKGPU2011/presentations/gnemo.pdf.

[14] Plamenka Borovska, Desislava Ivanova, "Code optimization and performance analysis of oceanographic software package NEMO for GPGPU systems." Computer Systems Department at the Technical University of Sofia. (2014). http://goo.gl/UyHW2m.

[15] Barcelona Supercomputing Center, "Marenostrum III (2013) system architecture." http://www.bsc.es/marenostrum-support-services/mn3, 2015.

APÉNDICE

A.1 Plataforma de ejecución - HPC

La ejecución del framework y el posterior análisis de resultados se ha llevado a cabo en el Supercomputador Marenostrum III ubicado en el Barcelona Computer Center (BSC-CNS). Las características de éste son las siguientes [15]:

- Pico performance : 1,1 Petaflops
- Memoria principal : 100,8 TB
- Nodos principales
  - 3.056 nodos de computo
  - 2x Intel SandyBridge-EP E5-2670/1600 20M 8-core at 2.6 GH
  - 64 nodos con 8x16 GB DDR3-1600 DIMMS (8GB/core)
  - 64 nodos con 16x4 GB DDR3-1600 DIMMS (4GB/core)
  - 2880 nodos con 8x4 GB DDR3-1600 DIMMS (2GB/core)
- Capacidad de almacenamiento: 2 PB
- Redes de comunicación
  - Infiniband FDR10
  - Gigabit Ethernet

A.2 Plataforma de ejecución - Aceleradores de cómputo

La ejecución del framework con aceleradores de cómputo y el posterior análisis de resultados se ha llevado a cabo en mi ordenador personal, con el siguiente dispositivo :

- Modelo : NVIDIA GeForce 710M
- Arquitectura : Fermi
- Memoria principal : 2048 MB
- Tamaño del warp : 32
- Numero máximo de threads : 1024