Universitat Autònoma de Barcelona

Departament d'Arquitectura de Computadors i Sistemes Operatius

Màster en Computació d'altes prestacions

Simulación de modelos orientados al individuo

Memoria del trabajo de ”Simulación de modelos orientados al individuo en sistemas multicores” del “Máster en Computació d'altes prestacions”, realizada por Marta Alberto Monferrer, bajo la dirección de Remo Suppi Boldrito Presentada en la Escuela de Ingeniería (Departamento de Arquitectura de Computadores y Sistemas Operativos)

2010

Iniciación a la investigación. Trabajo de fin de máster

Máster en Computación de Altas Prestaciones.

Título Simulación de modelos orientados al individuo en sistemas multicores

Realizada por Marta Alberto Monferrer
en la Escuela de Ingeniería, en el Departamento Arquitectura de Computadores y Sistemas Operativos

Dirigida por: Remo Suppi Boldrito

Firmado

Director Estudiante

Resumen

Los bancos de peces es uno de los más frecuentes grupos sociales en el reino animal. Muchas especies de peces se organizan en grupos sin la presencia de un líder. Muchos científicos atribuyen esta organización a dos patrones básicos de comportamiento; la atracción biosocial y la orientación paralela.

Este tipo de sistemas puede modelarse mediante la aproximación del Modelo orientado al Individuo, donde la conducta de cada individuo por separado, utilizando una serie de reglas, define el comportamiento grupal como la interacción de todos los individuos. En el caso de un banco de peces, el movimiento conjunto de los peces viene determinado por la elección individual de cada uno de ellos.

El desarrollo que ha experimentado la informática ha otorgado al modelo más capacidad de cálculo con la que dar respuesta en tiempos razonables a la simulación del modelo con mayores dimensiones o factores.

Por otro lado, aparecen los procesadores *multicore* en los sistemas de altas prestaciones, proporcionando un aumento del rendimiento gracias a la participación de nuevos *cores* de cómputo que se encuentran en un mismo procesador. Estos procesadores, al permitir que cada *core* tenga su propio hilo de ejecución, aumentan el paralelismo. El cambio de la arquitectura subyacente de los sistemas a *multicores* requiere de una revisión de las taxonomías de programación usadas a nivel de aplicación.

El objetivo que persigue el trabajo Simulación de modelos orientados al individuo en sistemas multicores es mejorar el rendimiento del simulador gracias a las ventajas que ofrecen los sistemas multicore, basándose en una programación híbrida.

Palabras Claves: Multicore, programación híbrida, Banco de Peces, Simulación, Modelo Orientado al Individuo, Computación de Altas Prestaciones.

1 Índice

2 Índice de figuras

3 Índice de tablas

Prólogo

Hasta hace unos años, el rendimiento de los procesadores ha ido aumentando a partir del incremento de la frecuencia de reloj del procesador. Sin embargo, aunque aún no se ha alcanzado el límite del rendimiento del reloj, cada año aumentando, pero a un ritmo menor, ya que se complica la disipación de la energía generada.

Es a partir de ese momento, tecnológicamente disruptivo, en el que aparecen los procesadores *multicores*, proporcionando el aumento del rendimiento esperado gracias a la participación de nuevos *cores* de cómputo que se encuentran en un mismo procesador. Estos procesadores, al permitir que cada *core* tenga su propio hilo de ejecución, aumentan el paralelismo. Cada *core* no sólo cuenta con su propia memoria caché L1, sino que, además, comparte con el resto de *cores* una memoria caché L2, de forma que los hilos que se ejecutan en paralelo pueden sacar partido de este espacio de memoria compartida a nivel de procesador.

El cambio de la arquitectura subyacente de los sistemas de altas prestaciones a *multicores*, consecuentemente, requiere de una revisión de las taxonomías de programación usadas a nivel de aplicación.

Es en ese ámbito en el que se enmarca el presente trabajo, en el que se estudiará la viabilidad, en un modelo biológico orientado al individuo, de aplicar taxonomías de programación, para obtener un mejor rendimiento gracias a la arquitectura de memoria compartida de los sistemas *multicores*.

El modelo orientado al individuo del banco de peces de *Huth and Wissel* [1,2] ha sido objeto de varios estudios en el Departamento de Arquitectura del computador y Sistemas Operativos (DACSO). Todas ellas tienen en común que las implementaciones se han realizado sólo teniendo en cuenta arquitecturas de sistemas con memoria distribuida.

En el presente trabajo se realizará una implementación del modelo del banco de peces de *Huth and Wissel* [1,2] con el objetivo de demostrar la mejora del rendimiento en sistemas multicores aplicando técnicas híbridas. La demostración de la mejora del rendimiento se efectuará a partir de experimentación.

En el capítulo de Introducción se realiza una puesto en contexto conceptual del problema que se aborda en el presente trabajo. En el capítulo de Modelado de Sistemas Biológicos se profundiza sobre el modelado de sistemas biológicos y se detallan los aspectos teóricos del modelo usado en el trabajo. En el capítulo de Implementación se desgranan las bases tecnológicas y matemáticas, junto con las consideraciones de diseño y rendimiento incorporadas en el simulador. En el capítulo Experimentación y resultados se puede comprobar los resultados obtenidos. Finalmente, en el capítulo Conclusiones se resumen las conclusiones y se sugieren líneas abiertas como trabajo futuro.

1 Introducción

El estudio y análisis del comportamiento de grupos de individuos existentes en la naturaleza es de mucho interés para los investigadores desde hace unos años. Investigaciones para comprender cómo y porqué los grupos se reproducen, mueven y conviven son cada vez de mayor relevancia, por lo importancia que ha cobrado hoy en día cualquier ayuda para la preservación de la biodiversidad de nuestro planeta. Muestra de ello es los trabajos de *Sato and Kitawaki* [15] simulación del cambio climático.

La ciencia computacional ayuda a afrontar este tipo de problemas complejos. A partir de un enfoque multidisciplinario, se aplican técnicas provenientes de cada una de las siguientes disciplinas: ciencia, matemática e informática.

La disciplina científica, ya sea biología, economía, etc, conoce el detalle del comportamiento del sistema real. A partir de las observaciones y del conocimiento de la disciplina científica, la matemática obtiene un modelo matemático. A partir de éste, la disciplina informática ya puede realizar la implementación del simulador. Una vez obtenido el simulador, debe realizarse una etapa de validación, en la que se comprobará que el modelo simulado cumple con los requisitos para los que se elaboró. Se trata de evaluar que el modelo se comporta de acuerdo a la definición recibida por parte de la disciplinas matemática. Una vez superada la etapa de evaluación, se deberá verificar el simulador, valorando las diferencias entre su funcionamiento y el sistema real que se estamos tratando de simular.

El concurso de las tres disciplinas es iterativo, con el objetivo de asegurar que el simulador se comporta lo más fielmente posible al sistema real. Las etapas de validación y verificación también ayudan a aumentar la fiabilidad del simulador. Esta aproximación iterativa y especializada para modelar la realidad nos ayudará a su interpretación y comprensión con mayor profundidad.

La simulación es la técnica con la que se puede reproducir el comportamiento de sistemas reales basándose en un modelo del sistema válido. De esta forma, será posible analizar y/o reproducir y/o predecir situaciones según cambios o variaciones que se provoquen de forma controlada en el sistema.

Por otro lado, la aparición de computadores paralelos ha promovido que las mejoras en las prestaciones de los simuladores crezcan a medida que crecían las prestaciones del *hardware*. Éstas prestaciones vienen acompañadas de mejoras en el *software*, de la mano de librerías específicas que permiten aumentar el aprovechamiento a través de las características de los computadores paralelos.

Por todo ello, la simulación ha ido adquiriendo cada vez más protagonismo en diversos ámbitos, científicos, e incluso comerciales, formando parte de procesos innovación o de

mejora de productos. Algunas investigaciones que implicaban un factor riesgo real (por ejemplo, prevención de incendios) o una escala de tiempo lejana (por ejemplo, cambio climático, predicciones de tiempo o de ventas) han sido reemplazados por la simulación.

Para el área de conocimiento tratada en este trabajo, la biología, existen dos enfoques para orientar el modelado de un sistema. Por un lado, está la alternativa del Modelado orientado a la Población, en la que el grupo se comporta como un todo. Por otro lado, está la opción del Modelo orientado al Individuo, en la que el comportamiento del grupo es resultado de la interacción entre todos los individuos.

Modelar un ecosistema mediante el enfoque del Modelado orientado al Individuo implica que se usa el concepto de entes individuales con identidad propia. Los individuos son entidades independientes que permiten modelar conceptos del mundo real y que caracterizan la aplicación que se está modelando. Cada individuo tiene un conjunto de valores o atributos que constituyen su estado interno. Un individuo específico sigue unos patrones de comportamiento que acaban reflejándose en su estado interno. Estas variables de estado lo convierten en un individuo singular frente al resto y pueden ir cambiando en el transcurso del tiempo.

Debido a que se guarda toda la información de estado a cada uno de los individuos del grupo, los Modelos orientados al Individuo requieren de una gran capacidad de cómputo, ya que la cantidad de información a tratar es muy elevada. Tal magnitud de información a manejar en Modelos orientados al Individuo conduce a tratar de conseguir mejoras en el rendimiento de la mano de las nuevas características de la arquitectura jerárquica de los sistemas *multicore.*

Los sistemas *multicore* están compuestos por nodos SMP, *Symmetric Multi-Processing.* Los nodos SMP *s*e caracterizan por el hecho de que tienen procesadores con varios *cores* independientes que comparten el acceso a la memoria. El nombre de simétrico viene no sólo por el acceso a la misma memoria compartida, sino también porque todos los *cores* ejecutan el mismo conjunto de instrucciones, aunque luego, según avance la ejecución, puede producirse una bifurcación, pero por lo menos todos los *cores* han lanzado la ejecución del mismo proceso. Los nodos SMP están unidos entre sí por una red de interconexión rápida.

Aplicando técnicas de programación paralela híbridas, que combinan el intercambio de datos entre nodos, junto con la paralelización, a nivel de procesador, de tareas o hilos que tienen acceso a la misma memoria compartida dentro del procesador del nodo, se supone que se puede lograr mejoras en el rendimiento.

La mejora del rendimiento de un Modelo orientado al Individuo en un ambiente paralelo en sistemas *multicore* es el objetivo del presente trabajo.

1.1 Problema

Este trabajo se enmarca dentro de la línea de investigación dedicada a la Simulación y modelos del comportamiento orientados al individuo que tiene el Departamento de Arquitectura del Computador y Sistemas Operativos (DACSO).

Los objetivos de este grupo son la investigación sobre técnicas de simulaciones de altas prestaciones en modelos orientados al individuo.

El modelo sobre el que se han venido realizando las investigaciones en el departamento es el modelo orientado al individuo del banco de peces de *Huth and Wissel* [1,2]. Se trata de un modelo biológico que es un ejemplo de auto-organización. El modelo cumple las siguientes hipótesis:

- Cada pez nada con el grupo de acuerdo al mismo modelo de comportamiento. Es decir, el grupo se mueve sin la presencia de un líder.
- No hay estímulos externos que afecten al movimiento
- En la velocidad y posición del pez, se incluyen influencias aleatorias
- El movimiento del pez sólo está influenciado por sus vecinos más cercanos
- El movimiento está compuesto por la fusión de las influencias de las siguientes conductas:
    - Repulsión (evitar chocar con un vecino que se encuentre a una determinada distancia vital)
    - Orientación paralela (respecto a los vecinos)
    - Atracción (intentar estar relativamente cerca de los vecinos)

El banco de peces presenta las siguientes características generales: por un lado, una fuerte cohesión (se hallan cercanos) y, por otro, un alto grado de polarización (se mueven en paralelo).

En el grupo de investigación del departamento ya se han atacado varios frentes de mejora de este modelo biológico. Desde la paralelización del modelo original, a la incorporación de tres dimensiones, e inclusive, la incorporación de estímulos externos al modelo. Pero en todos estos trabajos, las implementaciones se basaban en que la arquitectura del sistema de altas prestaciones sólo contaba con memoria distribuida. Para la paralelización del problema se utilizó la estrategia de fraccionar el espacio de datos a computar en procesos lógicos independientes y sincronizar, en serie después del cómputo, los datos dependientes entre procesos lógicos mediante mensajes.

Sin embargo, la realidad es que, actualmente, los actuales sistemas de altas prestaciones

ya emplean una arquitectura jerárquica con acceso a memoria distribuida y también compartida. Estos sistemas de altas prestaciones están compuestos de nodos SMP con varios procesadores *multicore* que disponen de una memoria compartida a nivel de procesador en los *cores* que la integran y de una red de interconexión rápida que permite que los nodos se comuniquen.

Ninguno de los trabajos realizados en el departamento ha sacado provecho de los actuales sistemas *multicore*, ésa es la principal aportación de este trabajo, analizar las posibilidades de paralelización del simulador del modelo biológico *Huth and Wissel* [1,2] para aumentar el paralelismo haciendo uso de la nueva arquitectura jerárquica que aportan. Puede explotarse, no sólo, como hasta ahora, el intercambio de datos entre nodos, sino también, la paralelización a nivel de procesador, de tareas o hilos que tienen acceso a la misma memoria compartida dentro del procesador del nodo, favoreciendo un acceso más óptimo a la memoria, y consiguiendo la reducción del tiempo total de ejecución debido a que se logra un acceso más rápido a los datos en los *cores* de un mismo procesador.

Para lograr un mejor aprovechamiento de los recursos de esta arquitectura jerárquica subyacente de los sistemas *multicore*s se ha de contar con la ayuda de nuevos paradigmas de programación. La programación híbrida es el nuevo paradigma que permite harmonizar la aplicación paralela con la arquitectura jerárquica de un sistema *multicore*, permitiendo obtener una mayor eficiencia en el rendimiento.

El objetivo planteado en este trabajo es aplicar técnicas de programación híbrida en el simulador de modelos orientados al individuo para sistemas *multicores* para mejorar su rendimiento a partir del paralelismo no sólo entre nodos, sino también a nivel de hilos en el procesador del nodo que tienen acceso a memoria compartida [13,16,18].

1.2 Objetivo

El objetivo que persigue el trabajo Simulación de modelos orientados al individuo en sistemas multicores es mejorar el rendimiento del simulador gracias a las ventajas que ofrecen los sistemas multicore, basándose en una programación híbrida. El desarrollo se mantendrá fiel al modelo matemático de simulación del comportamiento de un banco de peces de *Huth and Wissel* [1,2].

La programación híbrida es una programación paralela que permite combinar el intercambio de datos entre nodos, junto con la paralelización, a nivel de procesador, de tareas o hilos que tienen acceso a la misma memoria compartida dentro del procesador del nodo. Ayuda a harmonizar la aplicación paralela con la arquitectura jerárquica de memoria que se encuentran en un sistema *multicore* (memoria distribuida entre nodos y

compartida dentro del nodo). [13,16,18]

En la figura 1 se ilustran las taxonomías comúnmente aceptadas [13] como modelos de programación paralela híbrida:

Figura 1: Taxonomías programación híbrida

En la taxonomía híbrida *master-only* la comunicación entre procesos se efectúa fuera de la región paralela a través de mensajes MPI. OpenMP, facilita el acceso a las variables que se hallan ubicadas en la memoria compartida del procesador. No hay solapamiento de las comunicaciones con el cómputo. Es decir, se realizan en serie la sincronización de datos, si fuera necesaria, que se encuentran distribuidos en los nodos.

En la taxonomía *solapamiento comunicaciones-cómputo* la comunicación entre nodos se efectúa en paralelo al mismo tiempo que se realiza el cómputo en los hilos de región paralela con acceso a la memoria compartida dentro del procesador. Hay solapamiento de las comunicaciones con el cómputo. Es decir, se realiza en paralelo la sincronización de datos, si fuera necesaria, que se encuentran distribuidos en los nodos al mismo tiempo que se ejecutan mediante OpenMP varios hilos en paralelo que tienen acceso a la memoria compartida local del procesador.

Las características propias de cada problema a abordar junto con el entorno disponible serán factores decisivos considerar para tomar la decisión de cuál es la mejor estrategia a adoptar. A priori, no puede recomendarse una alternativa u otra sin haber realizado una experimentación rigurosa. Aunque sí es esperada conseguir una mejora del rendimiento ya sólo por estar utilizando regiones paralelas con acceso a memoria compartida en cada procesador del nodo.

Entre otras, cuestiones como el modelo matemático, la cantidad y estructura de los datos, la dependencia de los datos entre procesos, la arquitectura del sistema y la red de interconexión, afectan en el rendimiento de la aplicación.

Como objetivos específicos se han planteado:

- Decidir, a partir de la experimentación, qué taxonomía de programación híbrida proponer en el trabajo: *master-only* o *solapamiento comunicaciones-cómputo*
- Identificar técnicas de programación híbrida que mejoren el rendimiento
- Identificar parámetros de configuración propios del sistema *multicore* susceptibles de ser especificados según el contexto de ejecución para adaptar la ejecución al entorno.
- De estos puntos, presentar conclusiones y recomendaciones, basándonos en la experimentación

1.3 Antecedentes

Como se había introducido anteriormente, el grupo de investigación dedicado a la Simulación y modelos del comportamiento orientados al individuo ya ha realizado varios trabajos que evolucionan y mejoran el modelo biológico de *Huth and Wissel* [1,2]. Los principales son:

1. Primera implementación paralela [12] del modelo serie de *Huth and Wissel* [1,2]
2. Implementación paralela con mejoras de rendimiento a partir de las características espaciales de los datos [5]
3. Implementación paralela con mejoras de rendimiento y extensión tridimensional del modelo [4]
4. Implementación paralela con inclusión de estímulos externos [11]

La experimentación de estos trabajos se ha llevado a cabo en entornos con arquitecturas de memoria distribuida, pero ahora el departamento ya tiene disponible sistemas *multicore* con nodos con memoria compartida que permiten aplicar nuevos paradigmas en el simulador para mejorar el rendimiento y conducen a explotar el paralelismo a otro nivel mejor.

Por un lado, se plantea el reto de maximizar el aprovechamiento de la memoria compartida entre los *cores* que hay en cada procesador del nodo mediante alguna técnica de paralelización. La propia definición del modelo biológico ya promociona el uso de variables compartidas entre los *cores* de un nodo.

Precisamente, para hacer un uso extensivo de la memoria compartida que hay en cada procesador del nodo, se ha definido específicamente la especificación OpenMP [10]. Se basa en un modelo de ejecución *multi-threading* que favorece la localidad espacial de los

datos ya que todos los hilos pueden acceder a la misma dirección de memoria. De hecho, se ha convertido en el estándar de paralelización en arquitecturas con memoria compartida.

Por otro lado, también se debe resolver la dependencia de datos entre nodos que impone el modelo, comunicando datos que se hallan en memorias, distribuidas, de nodos que están conectados a través de una red de interconexión.

La especificación que facilita esta comunicación remota entre nodos es MPI (Message Passing Interface) [9] . MPI proporciona un control completo sobre los datos distribuidos entre los procesos y su sincronización.

La combinación de ambas técnicas de programación, memoria compartida en un mismo nodo y distribuida entre nodos (*extra-node*), se denomina programación híbrida.

En este sentido, en la literatura se pueden encontrar estudios que realizan comparativas de rendimiento para recomendar mejoras [10,11], concluyendo que, en la programación híbrida, se depende de la naturaleza del problema, los datos, sus dependencias y la arquitectura real del sistema *multicore*.

2 Modelado de Sistemas Biológicos

El estudio y análisis del comportamiento de grupos de individuos existentes en la naturaleza está recibiendo desde hace años un interés creciente. Investigaciones para comprender cómo y porqué los grupos se reproducen, mueven y conviven son cada vez de mayor relevancia, ya que es fundamental encontrar un equilibrio entre los intereses comerciales y la preservación de las especies.

La complejidad de los modelos biológicos se encuentra estrechamente ligada al nivel de abstracción del sistema que se quiere realizar en el modelado. En la simulación la complejidad esta en el número de individuos a simular y en la cantidad de factores considerados del ecosistema.

En el presente trabajo la complejidad se ha considerado en el número de individuos a simular. El sistema modelado utiliza un modelo básico del comportamiento del banco de peces. De esta forma, el trabajo se ha dirigido a aspectos más bien tecnológicos, centrándose en obtener una mejora del rendimiento del modelo sobre un sistema *multicore*.

La simulación es una herramienta fundamental para la resolución de un problema biológico o el estudio del ecosistema o parte de él, ya que no se puede contar con el sistema real. Por ello, en este trabajo se hace uso de un simulador que ayude a comprobar cómo se comporta el sistema.

El modelo utilizado en el presente trabajo, el banco de peces, es un modelo orientado al individuo que describe el movimiento de peces tomando como elemento fundamental el individuo. En realidad, el comportamiento del grupo es consecuencia directa de la interacción de los distintos integrantes del banco de peces.

El modelado aplicado a la biología debe tener en cuenta que cada individuo puede ser considerado como único, y se diferencia de los otros en fisiología y comportamiento, ya que depende de su composición genética y la influencia medioambiental. Es decir, de sus propiedades como individuo. Por ello, no se puede agregar es información dentro de una sola variable de estado.

Otro principio biológico a considerar es que un organismo dado es principalmente afectado solamente por otros organismos en su vecindad espacio-temporal.

En los modelos enfocados al individuo, el principio fundamental es que, a partir de las características de los individuos constituyentes, se derivan las propiedades del sistema biológico global.

2.1 Modelo orientado al individuo

El fundamento de los Modelos orientados al Individuo, MoI, es describir la conducta de cada individuo por separado, utilizando una serie de reglas, y obtener el comportamiento grupal como la interacción de todos los individuos.

En los modelos MoI la unidad es el individuo y las propiedades o características de la población se deducen a partir de la interacción de dichos individuos. Los atributos que definen el estado de cada individuo se mantienen actualizados a través del tiempo. Esto permite conocer la heterogeneidad de los individuos ya que los miembros de una misma especie se diferencian entre sí por los atributos que pertenecen a cada individuo.

El desarrollo que ha experimentado la informática ha otorgado a este modelo más potencia de cálculo con la que dar respuesta en tiempos razonables a complejos modelos y/o simular mayores dimensiones y factores del ecosistema.

La definición de los MoI se basa en reconocer a cada individuo común o un agente autónomo que actúa según un conjunto de reglas biológicas inherentes del sistema. El problema radica, evidentemente, en construir un conjunto de reglas realistas para organismos complejos.

El modelo de *Huth and Wissel* [1,2] sigue el modelo MoI. Basándose en la interacción de todos los individuos, se determina el movimiento de un banco de peces. A partir de la observación del comportamiento del grupo se puede observar que su movimiento figuras se rige por reglas básicas del comportamiento de los peces individuales.

Los bancos de peces es uno de los más frecuentes grupos sociales en el reino animal. Muchas especies de peces se organizan en grupos sin una estructura jerárquica. Muchos científicos atribuyen esta organización a dos patrones básicos de comportamiento; la atracción biosocial y la orientación paralela.

Estos bancos muestran un alto nivel de sincronización durante largos periodos de tiempo. Por tanto el movimiento de un banco de peces tiene una gran orientación paralela. El tipo de banco en el cual todos los individuos tienen orientación paralela es llamado banco polarizado. Ejemplos típicos de formación de grupos con orientación paralela sin la presencia de un líder son los *herring* o *mackerel* [20].

2.2 Modelo de Huth and Wissel

El modelo biológico implementado en el presente trabajo, es el desarrollado por *Huth and Wissel* [1,2].

La característica más relevante del modelo es su alto grado de cohesión en los movimientos grupales y que esta cohesión la tienen sin la presencia de un líder. El modelo original no tiene en cuenta la presencia de estímulos externos. El movimiento conjunto de los peces viene determinado por la elección individual de cada uno de ellos. Esta elección depende de la reacción a cómo se encuentran localizados físicamente sus vecinos. Es decir, cada pez observa el movimiento de sus peces vecinos con sus ojos y su línea lateral.

El modelo *Huth and Wissel* [1,2] se basa en las siguientes premisas:

- Cada individuo que conforma el grupo tiene el mismo patrón de comportamiento. Esto garantiza que el grupo se moverá sin presencia de un líder.
- El modelo del grupo no está afectado por influencias externas. Las influencias aleatorias son tomadas en cuenta para cada individuo. Por tanto, la posición y la velocidad de cada pez se basan en variables estocásticas.
- El movimiento de cada pez es influenciado por sus vecinos cercanos.

El movimiento de los peces se determina por un conjunto básico de patrones de comportamientos que indica cómo reacciona un pez al interactuar con otros individuos cercanos.

Cada pez en el banco cambia su dirección y su velocidad después de un paso de tiempo (iteración). Esta nueva dirección y velocidad depende de la posición, orientación y velocidad de un número fijo de peces vecinos. La influencia de un vecino depende de su distancia respecto al individuo a evaluar. Se distinguen cuatro influencias diferentes:

atracción, orientación paralela, repulsión y búsqueda.

A continuación se detallarán las características que determinan los movimientos de los peces en función de las influencias de sus vecinos.

En primer lugar, se tendrán en cuenta los peces vecinos que influenciarán en la reacción del pez. Dicha elección vendrá dada por aquellos que se encuentren a una cierta distancia, considerada próxima, y cuya posición sea visible para el pez en cuestión.

Si existen más de cuatro peces que cumplen las condiciones de cercanía y grado de visión, el pez escogerá aquellos cuatro vecinos que, o bien estén más próximos, o bien estén más en frente a su posición. La prioridad por frontalidad se basa en que primero se escogen aquellos peces que están por delante y más enfrente, para luego escoger los que están detrás y más alejados de su centro.

En la figura 2 se ilustra la relación de distancias, ángulos y zonas sobre los que el pez determinará cuáles son sus vecinos y que acabarán influenciando en su reacción, es decir, el movimiento resultante (nueva velocidad).

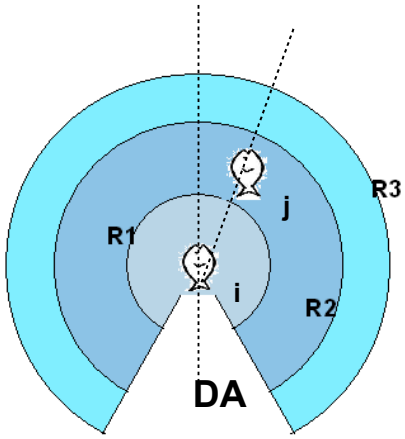

Figura 2: Tipos de influencia en la reacción de un pez

Como se puede apreciar en la figura 2, existirá un circulo, el mayor de todos, que marcará la zona de influencia del pez, tanto en distancia como en ángulo de visión. Dentro del mayor de los círculos habrá otros dos internos que servirán para dividir la zona de influencia del pez en tres partes: atracción, orientación paralela y repulsión. DA es la zona de área muerta sobre la que el pez no tiene visión.

Teniendo en cuenta que el pez está situado justo en el centro del circulo, se tendrá como primera zona aquella que está situada a una menor distancia que el radio R1 (tamaño del radio correspondiente a la mitad de un pez) y por tanto muy próxima al pez. Esta área es la de repulsión ya que ante la gran proximidad de un pez vecino, el pez tiende a apartarse para no colisionar. Para ello, el pez realizará un giro para estar perpendicular a la dirección del vecino con el que podría colisionar.

La siguiente zona, será la formada entre el radio R1 y el R2 (tamaño del radio correspondiente al doble del tamaño de un pez). Esta área es la de orientación paralela, ya que el pez tiende a ponerse en una posición paralela al pez vecino que está localizado en esa zona. El pez reacciona tomando el mismo ángulo que su vecino.

La última zona de influencia, más alejada, será la formada entre el radio R2 y el radio R3 (tamaño del radio correspondiente a cinco veces el tamaño del pez). Esta área es llamada de atracción, ya que el pez tiende a acercarse a su vecino. Por tanto el pez cambiará su dirección para ir en busca de su vecino.

Finalmente estará la reacción de búsqueda que se encuentra situada más allá del radio R3 y que se dará cuando el pez no perciba vecinos a su alrededor o bien porque están demasiado lejos o bien porque se encuentran en su ángulo muerto de visión, situado justo detrás, con un ángulo de 30 grados y que se llama DA o área muerta. En este caso, el pez lo que hará será modificar su actual dirección para intentar localizar a otros peces.

En función de las influencias de los vecinos, se consigue la nueva reacción del pez y consecuentemente su nueva dirección y posición. El cálculo final de la dirección vendrá dado como la media de las direcciones resultantes de las reacciones respecto cada vecino.

En un banco peces, la distancia con el vecino más cercano (Nearest Neigbour Distance, NMD), varia para cada especie pero tiene un valor promedio que se encuentra entre 0.3 y 3 veces el largo del pez (Body Lengths, BL). Donde los valores típicos de R1, R2, R3 son 0.5BL, 2BL, 5BL. DA es el área muerta en la que el pez no ve si hay algún vecino.

El pez reacciona de forma diferente de acuerdo al rango en el cual un vecino está posicionado. Se define r como la distancia del pez y su vecino. Los rangos están determinados por los radios R1, R2, R3. Los vecinos posicionados en el área de búsqueda no tienen influencia. Los peces se encuentran representados por las coordenadas de su posición P y por un vector velocidad v.

Las cuatro reacciones modeladas pueden ser divididas en dos grupos, uno estaría integrado por la reacciones de repulsión, atracción, y búsqueda de vecinos que son las reacciones que tienden a favorecer el alineamiento paralelo. El segundo grupo está conformado por la reacción de la orientación paralela que es la orientación a la que tiende el banco por naturaleza.

Orientación paralela

Si el i-enésimo pez tiene un j-enésimo vecino entre los radios R1 y R2, el i-enésimo pez nada en la misma dirección de su vecino, porque el j-enésimo vecino se encuentra en el área de orientación paralela. Esta reacción se ilustra en la figura 3 y se puede expresar

como: si, R1 < r < R2 y no se encuentra en DA, entonces el i-enésimo pez gira obedeciendo el ángulo de rotación del vector velocidad denotado como: $\beta_{ij} = \measuredangle(\vec{v_i}, \vec{v_j})$, donde v_i y v_j son los vectores de velocidad del i-enésimo pez

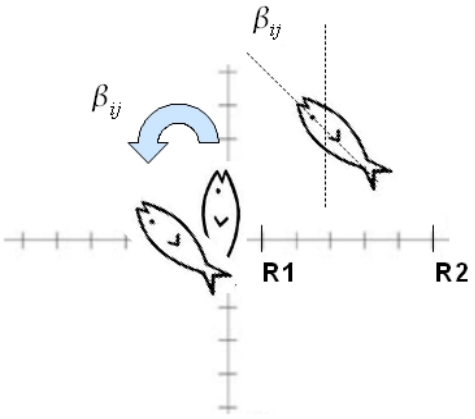

Figura 3: Reacción de orientación paralela

Repulsión

Este patrón de comportamiento está definido para evitar colisiones entre los peces. Durante el desplazamiento del banco de peces, la distancia entre los peces no es constante y pueden llegar a ser menores que R1 en algún momento. Si el i-enésimo pez tiene un j-enésimo vecino entre los radios R1 y 0, el i-enésimo pez gira para quedarse perpendicular a la dirección del vector velocidad de su vecino. Esta reacción se ilustra en la figura 4 y se puede expresar como: si, *r* < R1 y no se encuentra en DA, entonces el i-enésimo pez gira obedeciendo el ángulo de rotación del vector velocidad de la siguiente forma: $\beta_{ij} = min \measuredangle(\vec{v_i}, \vec{v_j}) \pm 90^o$

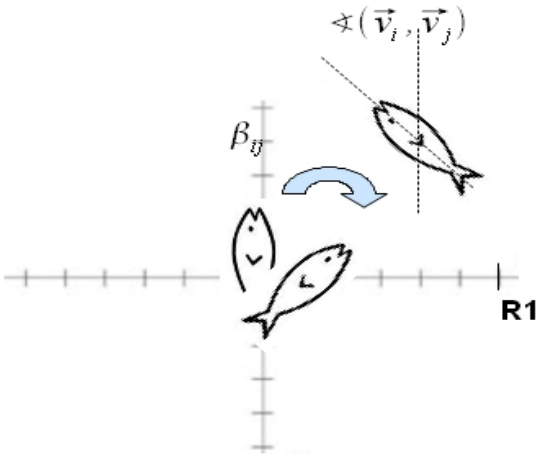

Figura 4: Reacción de repulsión

Atracción

Está demostrado que existe una atracción biosocial entre peces de la misma especie. Si un pez se aleja del banco, entonces, el pez nada hacia el grupo de nuevo. Esta reacción se ilustra en la figura 5 y se puede expresar como: si, R2 < r < R3 y no se encuentra en DA, entonces el i-enésimo pez gira en la misma dirección que el vector resultante de la diferencia de P_j y P_i obedeciendo la siguiente regla: $\beta_{ij} = \measuredangle(\vec{v}_i, \overrightarrow{P_j} - \overrightarrow{P_i})$

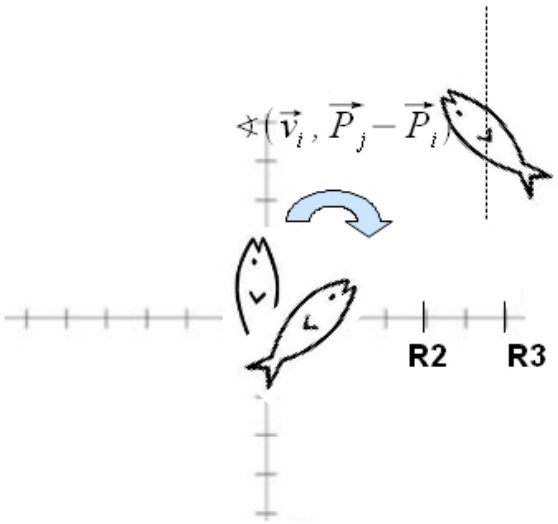

Figura 5: Reacción de atracción

Búsqueda de vecino

Si el pez vecino, j-ésimo, está muy lejos o dentro de la zona DA, el pez i-ésimo no puede verlo y comienza la búsqueda de algún pez vecino variando su ángulo en forma aleatoria. Esta reacción se puede expresar como: si, r > R3 o se encuentra en DA, entonces la reacción viene dada por la siguiente ecuación:

$$\beta_{ij} = chance([-180º, 180^o])$$

El cálculo de la posición y la velocidad de un vecino no es exacta por parte del i-ésimo pez, ya que cuenta con cierto grado de incertidumbre. Esto hace que el ángulo de rotación βij , que se calculó no sea el real. Para contemplar estas incertidumbres e influencias aleatorias, se construye una distribución de probabilidad p(αi) para el ángulo de rotación βij.

Para simplificar el modelo, se considera el modulo del vector velocidad v_i del i-enésimo pez independiente de los otros individuos. La velocidad es calculada al azar mediante una

distribución Gamma obtenida en base de datos experimentales. La distancia desplazada, al igual que los radios de las zonas de interacción de los peces, es medida en BL. Por tanto la velocidad promedio es de 1,2BL sec-1.

Lo más importante del modelo de *Huth and Wissel* [1,2] es la combinación de influencias de los vecinos cercanos en cada pez según las reacciones que se han descrito: atracción, reacción, orientación paralela y búsqueda de vecino.

3 Implementación

En el presente capítulo se desgranarán los detalles de la implementación del simulador. Incluirá las especificaciones del modelo descrito en el capítulo anterior del banco de peces *Huth and Wissel* [1,2] y detalles referentes a los modelos de programación paralela a aplicar.

Comprender en profundidad el modelo del banco de peces es fundamental para conseguir la mayor eficiencia posible con la programación híbrida. Cuestiones como la dependencia de datos y las necesidades de sincronización entre procesos paralelos vienen directamente impuestas por la propia definición del modelo.

El modelo puede implementarse mediante las siguientes taxonomías híbridas: la taxonomía *master-only* o con la taxonomía *solapamiento comunicaciones-cómputo*.

En la taxonomía híbrida *master-only* la comunicación entre procesos se efectúa fuera de la región paralela a través de mensajes MPI. OpenMP, facilita el acceso a las variables que se hallan ubicadas en la memoria compartida del procesador. No hay solapamiento de las comunicaciones con el cómputo. Es decir, se realizan en serie la sincronización de datos, si fuera necesaria, que se encuentran distribuidos en los nodos.

En la taxonomía *solapamiento comunicaciones-cómputo* la comunicación entre nodos se efectúa en paralelo al mismo tiempo que se realiza el cómputo en los hilos de región paralela con acceso a la memoria compartida dentro del procesador. Hay solapamiento de las comunicaciones con el cómputo. Es decir, se realiza en paralelo la sincronización de datos, si fuera necesaria, que se encuentran distribuidos en los nodos al mismo tiempo que se ejecutan mediante OpenMP varios hilos en paralelo que tienen acceso a la memoria compartida local del procesador.

En ambas, antes de seguir con el siguiente avance en el tiempo del simulador, debe realizarse una sincronización de los datos de los procesos lógicos de cada nodo.

A priori, no puede afirmarse que ninguna de estas taxonomías sea la mejor solución posible, sin haber efectuado previamente una fase de experimentación rigurosa. Por este motivo, se ha realizado un simulador flexible que contempla ambas taxonomías híbridas:

master-only y *solapamiento comunicaciones-cómputo*.

Será la experimentación, la técnica científica que permite que se pueda llegar a conocer cuál es la taxonomía de programación híbrida más adecuada para el modelo de un banco de peces *Huth and Wissel* [1,2] en el ambiente seleccionado.

3.1 Análisis y desarrollo

La computación de altas prestaciones intenta dar respuesta a numerosos problemas de Ciencia e Ingeniería, que requieren el procesamiento de grandes cantidades de datos numéricos, reduciendo los tiempos de ejecución, y facilitando la resolución de problemas de mayor complejidad.

La computación paralela emplea elementos de procesamiento simultáneamente para resolver un problema. Esto se logra dividiendo el problema en partes independientes de tal forma que cada elemento de procesamiento pueda ejecutar su parte del algoritmo a la misma vez que los demás.

Lorek and Soneschein [21] propusieron dos posibilidades para distribuir la simulación de un banco de peces: de forma estática o dinámica. La distribución estática consiste en asignar aleatoriamente peces a los procesos lógicos y esta distribución no cambiará durante la simulación. La distribución dinámica, en cambio, asigna peces a cada proceso lógico basándose en su posición. De esta forma, cada proceso simula una parte del espacio total. También en el mencionado trabajo [21] se recomienda que el uso de la distribución dinámica para grandes poblaciones, ya que los vecinos de un pez se mantendrán juntos en el mismo proceso o en uno vecino si migran.

En trabajos anteriores del departamento [4], [5] y [11] también se ha dividido la aplicación basándose en el espacio simulado. Por ello, en este trabajo también, para el simulador del modelo biológico del banco de peces, se efectuará una división del espacio en el que encuentran los peces según el número de procesadores y a cada procesador se le distribuirán los datos de sus peces.

Según la distribución de la memoria, las arquitecturas de los sistemas de altas prestaciones, pueden clasificarse en:

- Sistemas con memoria compartida. En ellos, todos los procesadores acceden a las mismas direcciones de memoria, es decir, los cambios en la memoria son visibles por todos. Tiene el inconveniente de ser caro, difícil de escalar y es complicado la programación de aplicaciones
- Sistemas con memoria distribuida. En ellos, cada nodo tiene su propia memoria local, la memoria local de un nodo no es visible por los demás nodos y la información es

compartida entre nodos por una red de interconexión. Los cambios en la memoria no tienen ningún efecto en el resto de nodos. Escala con facilidad, pero es difícil sincronizar estructuras de datos entre nodos

- Sistemas con memoria jerárquica, *multicore*. En este último grupo se encuentra el sistema objeto de este trabajo. En ellos, cada nodo tiene procesadores con varios *cores* que comparten una misma memoria, los nodos pueden comunicarse entre sí mediante una red de interconexión. Como dato del grado de aceptación de este tipo de arquitectura en el ámbito de las altas prestaciones, remarcar que el 85% de los sistemas del top500 [19] son *multicore* con procesadores *quad-core* y que el 5% ya tiene 6 o más *cores*.

Será un sistema *multicore* el que se considerará en este trabajo. En la figura 6 se ilustra la arquitectura jerárquica del sistema *multicore*.

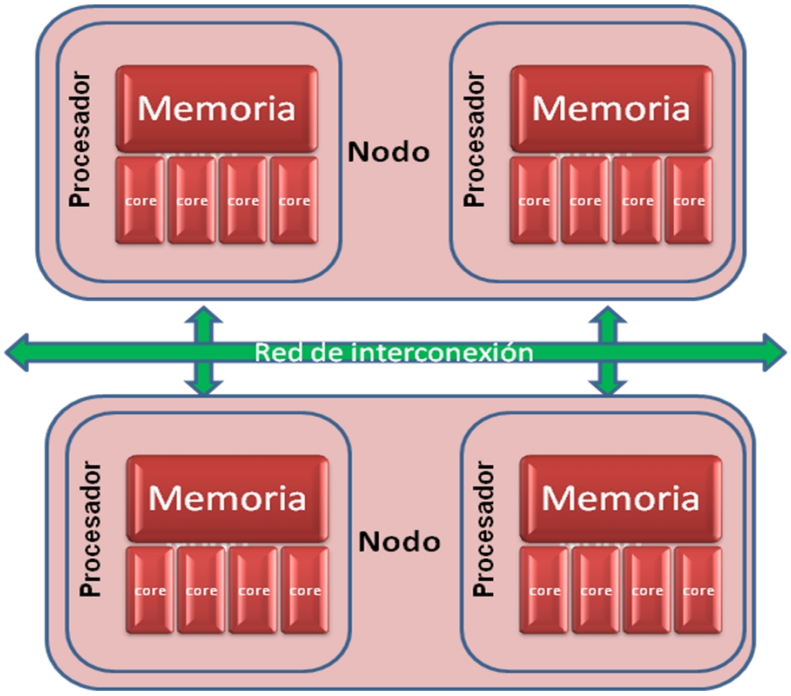

Figura 6: Sistema multicore

La programación híbrida es una programación paralela que permite combinar el intercambio de datos entre nodos, junto con la paralelización, a nivel de procesador, de tareas o hilos que tienen acceso a la misma memoria compartida dentro del procesador del nodo. Ayuda a harmonizar la aplicación paralela con la arquitectura jerárquica de memoria que se encuentran en un sistema *multicore* (memoria distribuida entre nodos y compartida dentro del nodo). [13,16,18]

En realidad, las características propias de cada problema a abordar junto con el entorno disponible son factores a considerar en la decisión de cuál es la mejor estrategia a adoptar. A priori, no puede recomendarse una alternativa u otra sin haber realizado una

experimentación rigurosa.

Entre otras, cuestiones como el modelo matemático, la cantidad y estructura de los datos, la dependencia de los datos entre procesos, la arquitectura del sistema de altas prestaciones y la red de interconexión, de cada problema afectan en el rendimiento de la aplicación.

En este sentido, en la literatura [6,15,16,17] se encuentran muchas comparaciones y análisis de rendimiento de programación híbrida versus la paralela pura, aplicadas a aplicaciones específicas. Aunque algunas demuestran una disminución en el rendimiento, son una minoría. En estas comparativas, para una aplicación específica, se evalúan los distintos modelos de programación híbrida, sobre distintas plataformas o aplicando diferentes cambios a la magnitud del problema. En el trabajo [17], para poder verificar cuál es el rendimiento de una aplicación en un sistema *multicore*, se evalúan las dos taxonomías híbridas que se implementarán en este trabajo. Por este motivo, y considerando que el modelo *Huth and Wissel* [1,2] requiere de varios accesos de consulta a los peces que se encuentran en la memoria de cada proceso lógico y que, además, se va a simular una población muy grande que puede suponer migraciones entre procesos lógicos, hace que se haya planteado la necesidad de implementar ambas taxonomías híbridas para poder verificar mediante la experimentación cuál es la más conveniente para el modelo.

Con la paralelización de la memoria compartida en cada procesador del nodo se consigue un paralelismo de grado fino y está diseñado para ser eficiente en sistemas SMP. Además, los códigos secuencial y paralelo son similares ya que se utilizan directivas para paralelizar.

La paralelización de memoria compartida se basa en que desde un proceso principal pueden crearse varios hilos. Esto permite tener múltiples caminos de ejecución concurrente. Cada hilo tiene sus propios datos, pero tiene la opción de compartirlos a nivel de proceso con el resto de hilos. Hay que tener en consideración que, la creación y gestión de estos hilos añade una sobrecarga de cómputo, por lo que, si no hay una cantidad relevante de memoria a compartir entre los hilos, el coste de la gestión de ese paralelismo puede restar efectividad al hecho de que se esté accediendo a memoria compartida.

En cambio, la sincronización de datos disponibles en otros nodos con la aplicación paralela es de grano grueso. En cada procesador se ejecuta una tarea que hace uso de su memoria local durante el cómputo. Para facilitar la sincronización entre nodos, los procesos intercambian datos a través de comunicaciones sobre una red de interconexión, enviando y recibiendo mensajes.

La programación híbrida permite fusionar los modelos de paralelización de memoria

compartida dentro de un procesador y acceso a datos distribuidos entre nodos. Por un lado, el espacio de direcciones compartido dentro de cada procesador del nodo es candidato a ser utilizado en la paralelización de memoria compartida a través de los hilos que se ejecutan. La API OpenMP permite la gestión de estos hilos. El paso de mensajes puede utilizarse como puente de comunicación entre nodos del sistema *multicore*. La API MPI ofrece las funciones necesarias para ello.

3.2 Modelo matemático Huth and Wissel

El trabajo desarrollado en [4] que incorpora la extensión tridimensional del modelo de *Huth and Wissel* [1,2] ha sido incluido en el presente simulador. La consecuencia matemática resultante fue el representar los vectores de posición y velocidad con una coordenada 3D respectivamente. Esto provoca que, geométricamente, el área de influencia sea una esfera, no circunferencia.

Para obtener la siguiente posición de un pez, es necesario seleccionar los vecinos más cercanos que le influencian y, de cada uno de ellos, aplicar a la velocidad el tipo de influencia que ejerce sobre él. Las influencias que se pueden aplicar sobre un pez son las siguientes: atracción, repulsión y orientación paralela.

3.2.1 Atracción

El pez siempre intenta nadar con el grupo. Este comportamiento provoca que el pez intente acercarse a los peces más cercanos.

Esto genera un nuevo vector de velocidad definido por la diferencia de las posiciones de los dos peces vecinos. La nueva direccion del i-ésimo pez respecto a su j-ésimo pez vecino viene dada por la siguiente fórmula: $\vec{v}_{ij} = \overrightarrow{P_j - P_i}$

3.2.2 Repulsión

El pez procura nadar a una cierta distancia vital de sus vecinos. Si esta distancia se hace más pequeña que un cierto umbral, el comportamiento del pez es de repulsión.

En la repulsión el i-ésimo pez trata de cambiar su dirección perpendicularmente respecto al j-ésimo pez. La velocidad de repulsión se logra solucionando las siguientes ecuaciones:

$\vec{v}_{ij} = n * \vec{v}_j - v_i$ y $\vec{v}_{ij} * \vec{v}_j = 0$

donde:

$$\alpha = \left| \measuredangle (\vec{v}_i, \vec{v}_i) \right| \text{ y } n = \left| \vec{v}_i \right| / \left(\left| \vec{v}_j \right| * \sin(\pi - \alpha) \right)$$

3.2.3 Orientación paralela

El pez intenta mantener la velocidad de nado en la misma dirección que la de sus vecinos. Este comportamiento individual provoca que el movimiento de grupo de un banco de peces sea altamente polarizado.

El i-ésimo pez intenta mantener su velocidad de nado hacia la misma dirección que su j-ésimo vecino. Esto implica que el cambio de orientación del pez i respecto al pez vecino j viene dado por la siguiente ecuación: $\vec{v}_{ij} = \vec{v}_j$

3.3 Memoria compartida: OpenMP

3.3.1 Introducción

OpenMP ("Open specifications for Multi Processing") [10] es una API, o interfaz de programación de aplicaciones, para la paralelización de programas en arquitecturas de memoria compartida. Su especificación es revisada y actualizada por una organización independiente constituida por importantes empresas de hardware y software, *el OpenMP Arquitecture Review Board*. Al ser una API, se independiza al usuario final de los detalles de la implementación que hay detrás de la API y aporta la ventaja de que el mismo código puede funcionar en diferentes plataformas, sólo se requerirá compilar de nuevo al cambiar de entorno.

OpenMP consta de tres grupos de herramientas de paralelización:

- Directivas de compilador para indicar el paralelismo al compilador. Mediante las directivas o pragmas se puede controlar cómo aplicamos el paralelismo a nivel de hilos. Estos pragmas están diseñados de forma tal que, aunque el compilador no los soporte, el programa seguirá teniendo un comportamiento correcto, pero sin paralelismo.
- Librería de funciones. De esta forma, en tiempo de ejecución, se puede consultar o modificar parámetros paralelos de OpenMP como número de los hilos que van a participar y el número de cada hilo.
- Variables de entorno. Permiten definir en tiempo de ejecución parámetros del

sistema en paralelo tales como el número de hilos.

OpenMP se aplica a un bloque estructurado de código. Se basa en el modelo fork-join de multi-threading, en el que una tarea se divide en K hilos (*fork*), para luego recolectar sus resultados al final y unirlos en un solo resultado (*join*). Cada hilo es asignado a un *core* del procesador. Es la API candidata ideal para modelos SPMD (Single Process Multiple Data).

Las dos alternativas posibles de paralelismo en OpenMP es, o bien, a nivel de bucle, o bien, a nivel de regiones paralelas. Ambas alternativas se definen con directivas en las que pueden indicarse valores por defecto de comportamiento, configuración y qué variables compartidas deben considerarse entre los hilos que se creen a continuación.

Los *cores* ejecutarán instrucciones diferentes en paralelo, pero tienen acceso al mismo espacio de direcciones de memoria. La posibilidad de compartir variables, su dirección de memoria, entre hilos paralelos es lo que favorece el paralelismo al permitir la optimización del uso de la memoria compartida de ese procesador por parte de todos sus *cores*. Sin embargo, esta gran ventaja provoca que surja un inconveniente: puedan producirse conflictos en los accesos de escritura a los datos que se deben proteger con sincronizaciones. Estas sincronizaciones son costosas en cuanto a tiempo de computación y deben minimizarse.

En la alternativa de paralelismo a nivel de bucle, iteraciones individuales del bucle principal, sin repeticiones, son distribuidos paralelamente en cada uno de los hilos. A medida que cada hilo acaba una iteración, se le asigna una nueva iteración a procesar en paralelo al resto de los hilos. A esto se le llama paralelismo de granularidad fina.

En la alternativa de paralelismo de regiones paralelas, cualquier sección del código puede ser paralelizada, aunque no por bucles. El trabajo es distribuido explícitamente en cada uno de los hilos usando el único identificador para cada hilo o mediante directivas específicas. También es llamado paralelismo de gruesa granularidad.

Una directiva OpenMP implica que se incluye una sincronización obligatoria en todo el bloque que viene a continuación. Es decir, el bloque de código se marcará como paralelo y se lanzarán hilos según las características que indique la directiva. Al final de la ejecución del código de la directiva habrá una barrera implícita que sincroniza los diferentes hilos.

En la Figura 7, se ilustra la diferencia de ambas alternativas de paralelización con OpenMP:

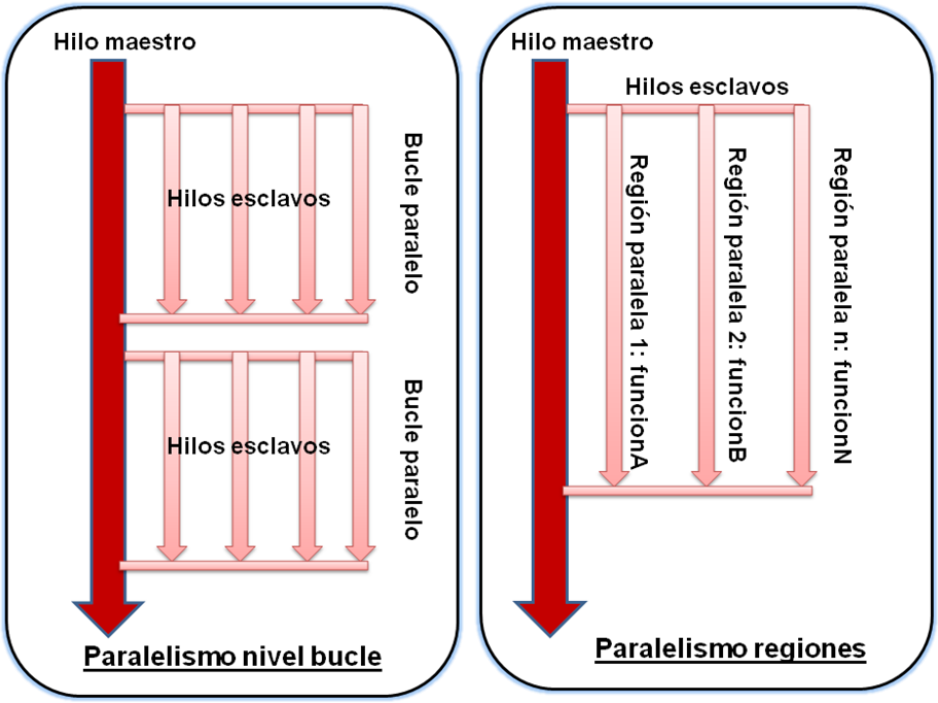

Figura 7: Alternativas paralelismo en OpenMP

Como puede deducirse, de las explicaciones anteriores sobre los fundamentos de OpenMP, se trata de una API sencilla pero poderosa que permite incorporar paralelismo en un nodo con procesadores con múltiples núcleos.

Sin embargo, OpenMP no es excesivamente flexible. El principal inconveniente que tiene es que está pensado fundamentalmente para el paralelismo de bucle. Aunque se pueden crear regiones paralelas, el hacerlo dinámicamente no es una tarea tan sencilla como con los bucles. Este es uno de los retos con los que hemos enfrentado en este proyecto, ya que deseamos verificar el comportamiento del simulador con la taxonomía *solapamiento comunicaciones-cómputo,* que requiere de la ejecución en paralelo de regiones especializadas: por un lado un hilo especializado en comunicaciones y el resto de hilos del nodo para cómputo.

3.3.2 Estrategias

En el presente apartado se explican las estrategias que se han seguido en el diseño e implementación del simulador en lo que se concierne a la API OpenMP.

La primera consideración que se ha seguido es utilizar sólo directivas de OpenMP. Aunque, como se ha indicado anteriormente, OpenMP sí dispone de una librería de funciones para C/C++ no se ha hecho uso de ella. De esta forma, se independiza el

simulador de la disponibilidad o no de la librería de memoria compartida. Se delega al momento de la compilación si debe incluirse la paralelización a nivel de memoria compartida. El mismo código es válido tanto para aplicaciones paralelas como para puras o híbridas.

La estrategia de independizar también se ha aplicado a la arquitectura subyacente. Tampoco se indica por variables de entorno cuál es el número de hilos. Se delega la configuración del funcionamiento a la propia API según la arquitectura subyacente del sistema *multicore*. OpenMP es capaz en tiempo de ejecución de saber cuántos hilos puede gestionar realmente en ese proceso (cores disponibles en el nodo o en el procesador). Aunque el número de hilos también puede indicarse dentro de los pragmas, tampoco se ha usado esta técnica para fijar el número de hilos. Así se consigue portabilidad del simulador a cualquier otro entorno.

En cuanto a las directivas de paralelización, la que activa el paralelismo a nivel de bucle, tiene una serie de opciones de planificación de las ejecuciones paralelas que también se han evaluado para poder recomendar cuál es la configuración más efectiva en el caso del presente trabajo con el modelo *Huth and Wissel* [1,2]. Hay tres tipos de planificaciones en las ejecuciones de los bucles:

- Estática: cada hilo decide independientemente qué pedazo del bucle debe procesar.
- Dinámica: no hay un orden predecible en el que los elementos del bucle son asignados a los diferentes hilos
- Guiada: produce un comportamiento entre la estática y la dinámica.

De estos tres tipos de planificación, la que ha resultado más recomendable, por el tiempo de ejecución obtenido, es la estática. Es la que la API OpenMP usa por defecto, si no se indica lo contrario.

Hasta ahora se ha estado detallando estrategias que tienen que ver con la API de paralelizacion exclusivamente, pero ¿cuál es el lugar más adecuado para poner las directivas de OpenMP?

Como puede deducirse de los apartados anteriores, relativos al modelo biológico de *Huth and Wissel* [1,2], hay dos factores claves que ayudarán a identificar cuál es el lugar idóneo para incorporar la paralelización mediante OpenMP:

- Bucles: iteraciones en el modelo para ejecutar operaciones de cómputo intensivo que sean susceptibles de dividirse entre los *cores* del procesador
- Memoria: que esas iteraciones necesiten acceder a variables de memoria y que puedan ocupar mucho tamaño o ser muchas: así se aprovecha la caché

compartida del procesador a nivel L2

Hay dos tareas en el modelo de *Huth and Wissel* [1,2] que cumplen ambas condiciones:

- Cuando se aplica el modelo de *Huth and Wissel [*1,2] a todo el banco: se itera por todo el banco de peces para aplicar el modelo a cada uno: búsqueda vecinos, selección vecinos, cálculo de velocidad, cálculo de posición
- Cuando, de cada pez, se buscan cuáles son sus vecinos que le influencian: se trata de un bucle que itera por todos los peces del banco de peces calculando distancias

En esas dos tareas se ha incluido la paralelización a nivel de bucle.

Adicionalmente, hay otras estrategias, recomendadas en la literatura [16,22], más comunes a considerar en un entorno *multi-threading*, pero no menos importantes, que también se han aplicado:

- Minimizar el número de regiones críticas: en un entorno *multi-threading* esto podría acarrear mucha penalización en el rendimiento. Esto es debido a que todos los núcleos quedan a la espera de que se acabe la ejecución de la región crítica, por lo que el tiempo de ejecución aumenta
- Eliminar cómputo y llamadas a las regiones críticas
- Minimizar el uso de barreras de sincronización en OpenMP (también aplicable a MPI)
- Verificar que el paralelismo está aplicado correctamente: en bucles con cómputo intensivo y que soliciten acceso a variables que estén en memoria compartida
- Minimizar el uso concurrente de recursos compartidos que requieran de mecanismos de exclusión mutua
- Eliminar dependencias del entorno de ejecución (del número de hilos o de cuál es mi número de hilo)
- Minimizar el tiempo en el que no hay regiones paralelas, ya que eso significa que sólo se está utilizando uno de los cores disponibles del procesador: no se usa eficientemente la arquitectura del sistema *multicore*

3.4 Paso de mensajes: MPI

3.4.1 Introducción

MPI ("Message Passing Interface") [9] es una especificación que permite comunicar datos entre procesos mediante el intercambio de mensajes sobre una red de interconexión.

MPI es una especificación estándar y portable, mantenida y evolucionada por el MPI Forum, que consta de unas 40 organizaciones participantes.

MPI está diseñado para desarrollar aplicaciones SPMD (Single Process Multiple Data). Al arrancar una aplicación se lanzan en paralelo N copias del mismo proceso. Estos procesos no avanzan sincronizados instrucción a instrucción sino que la sincronización, cuando sea necesaria, tiene que ser explícita a nivel de programación con las funciones adecuadas de la API MPI. Los procesos tienen un espacio de memoria completamente separado entre sí. El intercambio de información, así como su sincronización, se hacen mediante el paso de mensajes.

MPI dispone de funciones de comunicación punto a punto (sólo entre dos procesos), y de opciones colectivas (entre múltiples procesos). Gracias a ella se puede solventar la sincronización de datos en entornos con memoria distribuida.

3.4.2 Estrategias

En las comunicaciones entre los nodos adyacentes del simulador se han aplicado las siguientes estrategias, por un lado, comunicaciones no bloqueantes para facilitar que los datos se vayan tratando a medida que vayan llegando al nodo y, por otro lado, se ha aplicado un cierto grado de granularidad en las comunicaciones, es decir, no todo el bloque de datos de golpe.

Comunicaciones no bloqueantes

La especificación MPI permite una gran versatilidad en la comunicación entre procesos, pudiendo indicar cómo deben comportarse las funciones de envío. Hay dos alternativas: de forma bloqueante o no bloqueante. Esto permite incorporar una nueva mejora en el rendimiento global del simulador.

Por supuesto, el poder incorporar un comportamiento bloqueante o no bloqueante en las comunicaciones, dependerá del modelo del simulador y cómo éste impone que los datos dependientes estén sincronizados.

El beneficio del uso de las comunicaciones no bloqueantes se encuentra en que permite el

desacoplo entre las operaciones de lectura y escritura. De esta forma, el proceso que envía no tiene que esperarse a que el proceso de recepción realice la recepción efectiva.

Al desacoplar, y no bloquear, el proceso que ha realizado la solicitud de envío no tiene que esperarse y puede seguir realizando cómputo en su nodo independizándose de cuándo el proceso de recepción está listo para recuperar los datos del mensaje. El mensaje enviado es guardado en un buffer intermedio de la implementación MPI hasta que el proceso receptor pregunta si hay algún mensaje pendiente de serle entregado.

En el diseño de la implementación del simulador de este proyecto se ha considerado el uso de las comunicaciones no bloqueantes. El detalle técnico del envío no bloqueante es sencillo: se realiza mediante la operación MPI_Isend de MPI. En cuanto a la recepción, en el proceso destino del mensaje, deben realizarse los siguientes pasos:

- Preguntar si hay algún mensaje disponible a través de la operación MPI_iprobe
- Si la respuesta es que sí, recuperar el mensaje del *buffer* mediante la operación MPI_Irecv

Para las dos taxonomías híbridas de las que consta el simulador se ha aplicado la técnica de comunicaciones no bloqueantes, ya que no hay dependencia de datos durante el cómputo con el proceso lógico adyacente.

Granularidad en las comunicaciones

El comportamiento *multi-threading* proporcionado por la API OpenMP conduce al planteamiento de las comunicaciones de forma granular, aprovechando el efecto del cómputo en paralelo de cada uno de los hilos. No es necesario esperar a tener el cómputo de todos los hilos, sino que puede hacerse por etapas, favoreciendo el solapamiento de cómputo y comunicaciones de nuevo.

Cada hilo de cómputo se estará encargando del sub-conjunto de los peces que le correspondan al nodo. A priori, no tienen porqué tardar lo mismo, un hilo puede acabar su cómputo antes que el resto y su rango de datos ya está listo para ser enviado a los nodos adyacentes, si aplica.

Es en este punto donde se ha incorporado la granularidad en las comunicaciones. Los hilos van marcando sus datos como listos para enviar y el proceso de comunicaciones paralelo los verifica y envía los que corresponda a los procesos adyacentes.

Las comunicaciones se realizan por etapas, independizando los envíos y recepciones de cada uno de los hilos del nodo. Con esta estrategia, a medida que se calcula el modelo del conjunto de peces que corresponde a cada hilo, el hilo que está gestionando las comunicaciones se encargará de enviar los peces que hayan migrado y/o los que influyan

en cada uno de los procesos adyacentes del nodo actual. Se van enviando en bloques hasta que se ha completado el cálculo de los peces de ese hilo y por último se envía un mensaje de finalización de ese hilo.

De esta forma, se consigue una granularidad dinámica, adaptando los envíos y recepciones a la velocidad de cómputo de los diferentes hilos, logrando que el solapamiento de las comunicaciones sea más efectivo, al no tener que esperar a que todos los hilos de ese nodo hayan terminado su cómputo.

Esta estrategia sólo puede aplicarse para la taxonomía de *solapamiento comunicaciones-cómputo,* ya que contempla un hilo especializado para las comunicaciones. Es el hilo que está atento a que haya elementos a enviar de cualquiera de los hilos restantes del procesador.

En cambio, para la taxonomía de *master-only,* la información a enviar se empaqueta en un bloque entero y no se aplica ninguna técnica de granularidad. Esta alternativa de comunicaciones es la que se ha utilizado en los proyectos anteriores del departamento sobre el modelo del banco de peces *Huth and Wissel* [1,2].

Adicionalmente, se ha configurado la librería que implementa MPI, OpenMPI [23], para hacer un uso más adecuado de la arquitectura subyacente. OpenMPI permite indicar el número de nodos, procesadores y cores por procesador y dirigir la ejecución de los diferentes procesos al elemento de la arquitectura que más interese: nodo, procesador o core, consiguiendo una afinidad de procesos lógicos según la arquitectura real del sistema. Incluso puede indicarse si los procesos debe ir consecutivos (según la arquitectura *hardware*) o individualmente dónde va cada uno: favoreciendo que las comunicaciones puedan realizarse desde elementos afines según la topología real.

Mediante esta opción, se han lanzado los experimentos del simulador paralelo puro contra cada nodo independientemente y los experimentos de los simuladores híbridos a cada uno de los procesadores de cada uno de los nodos.

3.5 Implementación paralela

Una vez detallados el modelo matemático, y las técnicas de programación híbridas a emplear, se describe a continuación cómo se ha realizado la implementación híbrida del simulador paralelo.

Como primer aspecto a desarrollar, se detallará cuál es el funcionamiento serie del modelo biológico de *Huth and Wissel* [1,2]. En la figura 8 se muestran las etapas a realizar en cada iteración del simulador sobre cada uno de los peces.

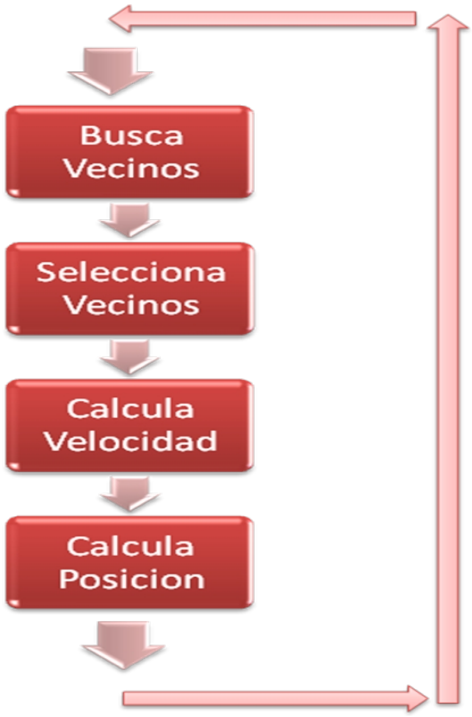

Figura 8: Diagrama flujo simulador serie Huth and Wissel

A continuación se describen las principales tareas que realiza el simulador serie:

- Búsqueda de vecinos: los candidatos válidos son los individuos que se encuentran dentro del radio de influencia y fuera del cono de visión ciega.
- Selección de los vecinos: según *Huth and Wissel* [1,2], sólo son significativos los cuatro vecinos que se encuentran más cercanos al frente. Si hubiera menos de cuatro candidatos, todos se considerarán vecinos.
- Calcula velocidad: según la distancia y velocidad de los vecinos seleccionados, el pez reaccionará. La suma de cada una de las reacciones individuales a cada uno dará una velocidad nueva de reacción del pez según el entorno actual.
- Calcula posición: con esta nueva velocidad, fruto de la postura del individuo frente al entorno, se calcula la nueva posición del pez dentro del banco de peces.

Como aspecto a destacar del algoritmo paralelo, es que es necesario consultar continuamente variables que se encuentran en memoria. La búsqueda de vecinos es la tarea que necesita acceder a las posiciones de memoria donde se encuentran los datos de todos los peces del banco.

Antes de empezar con la simulación, deben inicializarse los datos del banco de peces. Los individuos del simulador son peces cuya principal característica es que se encuentran uniformemente distribuidos en el espacio de simulación.

Una vez comprendido cuál es el algoritmo serie, podemos proceder a paralelizarlo. Como se ha descrito anteriormente, existen dos aproximaciones para distribuir los individuos en un simulador MoI: una dinámica y otra estática. En la estática, la distribución no cambia durante el tiempo. En la aproximación dinámica, cada individuo es asignado a un proceso lógico según su posición espacial. Ésta última es la alternativa adoptada en este trabajo, como también se hizo en [4], [5] y [11].

En el presente trabajo se han dividido los datos de la aplicación según el espacio simulado. El eje de división ha sido el X. En la figura 9 se ilustra esta estrategia de división de los datos.

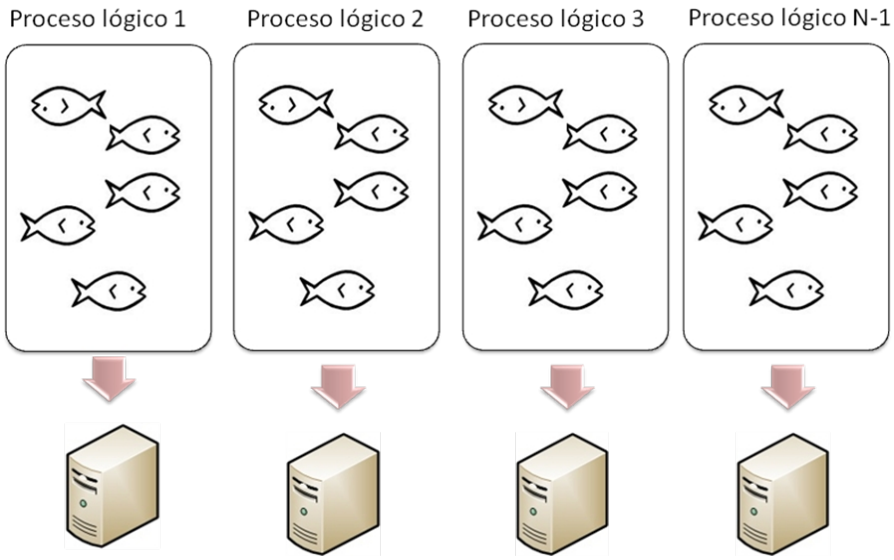

Figura 9: Asignación trabajo a los procesos lógicos

Se ha de tener en cuenta que los procesos no trabajarán aisladamente de los otros. Algunos procesos lógicos adyacentes necesitarán comunicarse entre sí, ya que hay puede haber peces que se encuentren cerca del borde y esté influyendo en los peces del proceso lógico adyacente (anterior o posterior). Los datos de posición y velocidad de estos peces cercanos, pero que se encuentran en otro proceso lógico, deberán ser comunicados al proceso lógico adyacente al que influencian.

En la figura 10 se puede ver gráficamente esta influencia entre procesos lógicos adyacentes.

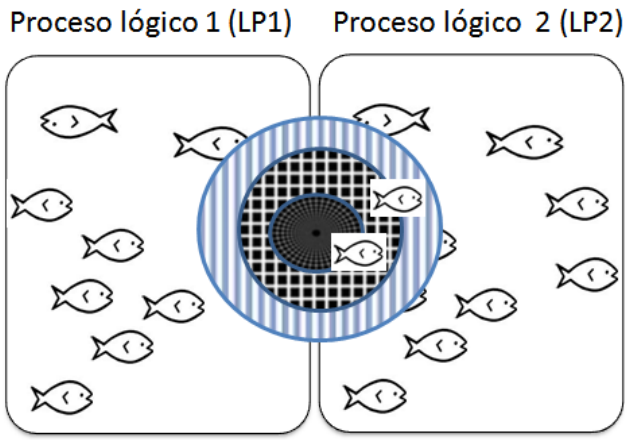

Figura 10: Influencia procesos lógicos adyacentes

Otra condición que provoca que haya comunicaciones entre procesos lógicos es cuando un pez migra de proceso lógico. Esto sucede cuando su nueva posición ya no corresponde al rango de posiciones que computa el actual proceso lógico. Cuando sucede esto, deben comunicarse, al nuevo proceso que gestionará los datos el pez, la información de posición y velocidad. Además, ese pez debe ser borrado del proceso lógico del que ha migrado.

Las comunicaciones en este caso también son entre procesos lógicos adyacentes: el proceso lógico LP2 sólo se comunicará con el LP1, cuando haya migraciones a la izquierda o peces en el borde izquierda, o con el LP3, cuando haya migraciones a la derecha o peces en el borde derecha que estén en el radio de influencia.

Implementación paralela MPI puro

La primera implementación paralela a realizar es la MPI puro. Aunque esta implementación ha sido la aproximación de los anteriores trabajos realizados sobre el modelo de *Huth and Wissel* [1,2] es la que se desea utilizar como referencia para demostrar, comparando tiempos de ejecución, que con una implementación paralela híbrida se puede mejorar el rendimiento del simulador.

Presenta una primera etapa de inicialización de datos, en la que cada proceso lógico recupera, de un único fichero, los datos que le corresponden del banco de peces. Serán los peces que están en el intervalo del espacio de simulación y los de los procesos adyacentes que le pueden influenciar.

A partir de ese momento, hasta que se haya avanzado tantas veces en el tiempo, como tenga configuradas el simulador, cada proceso lógico realiza las siguientes tareas:

- Etapa de cómputo: Por cada pez que gestiona el proceso lógico, se realizan las

siguientes tareas (en serie dentro del procesador del nodo):

- Búsqueda de sus vecinos entre todos los peces del proceso lógico
- Selección de los cuatro vecinos más cercanos, aplicando los criterios explicados en el apartado 3.2
- Cálculo de la nueva velocidad del pez como reacción a la influencia de los vecinos seleccionados, aplicando los criterios explicados en el apartado 3.2
- Cálculo de la nueva posición según la nueva velocidad y la antigua posición, aplicando los criterios explicados en el apartado 3.2

- Etapa de comunicaciones: Una vez procesados todos los peces, se procede a realizar las operaciones de comunicaciones, todas mediante mensajería MPI (en serie después de la etapa de cómputo):
  - Enviar a los procesos adyacentes los peces migrados y los datos de los peces influyentes
  - Recibir datos de los procesos adyacentes sobre peces migrados al proceso lógico y los datos de los peces que influencian al proceso lógico

En la figura 11 se puede observar el diagrama de flujo de la implementación paralela MPI puro.

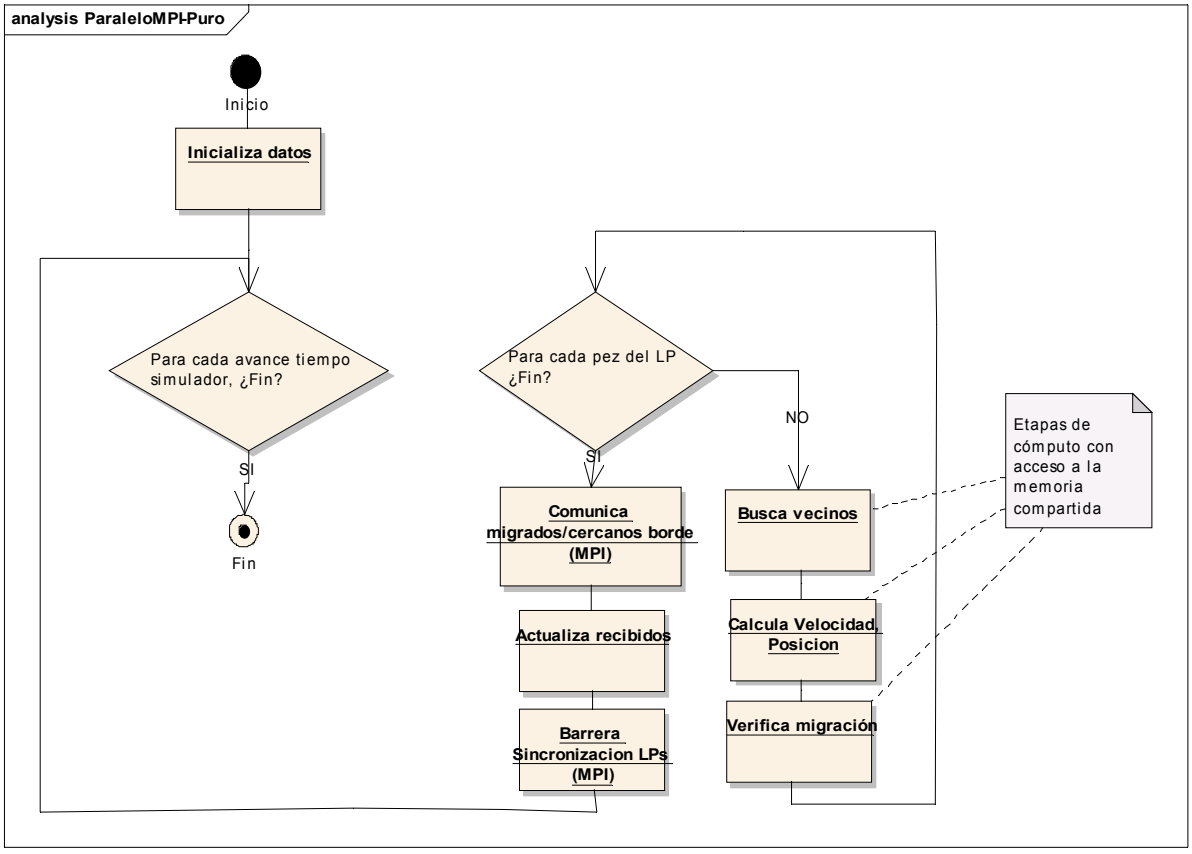

Figura 11: Diagrama flujo simulador paralelo MPI-puro

Implementación paralela híbrida *master-only*

Una vez realizada la implementación paralela MPI puro del simulador, se procede a implementar la híbrida *master-only,* que es la que tiene menos dificultad.

También presenta una primera etapa de inicialización de datos, en la que cada proceso lógico recupera, de un único fichero, los datos que le corresponden del banco de peces. Serán los peces que están en el intervalo del espacio de simulación y los de los procesos adyacentes que le pueden influenciar.

A partir de ese momento, hasta que se haya avanzado tantas veces en el tiempo, como tenga configuradas el simulador, cada proceso lógico realiza las siguientes tareas:

1. Etapa de cómputo: Las iteraciones sobre los peces del proceso lógico se realizan en paralelo en todos los *cores* del procesador del nodo mediante OpenMP. Por cada pez que gestiona el proceso lógico, se realizan las siguientes tareas:
    - Búsqueda de sus vecinos entre todos los peces del proceso lógico
    - Selección de los cuatro vecinos más cercanos, aplicando los criterios explicados en el apartado 3.2. En esta etapa, ya que es un bucle, también se aplica OpenMP
    - Cálculo de la nueva velocidad del pez como reacción a la influencia de los vecinos seleccionados, aplicando los criterios explicados en el apartado 3.2
    - Cálculo de la nueva posición según la nueva velocidad y la antigua posición, aplicando los criterios explicados en el apartado 3.2
2. Etapa de comunicaciones: Una vez procesados todos los peces, se procede a realizar las operaciones de comunicaciones, todas mediante mensajería MPI (en serie después de la etapa de cómputo):
    - Enviar a los procesos adyacentes los peces migrados y los datos de los peces influyentes
    - Recibir datos de los procesos adyacentes sobre peces migrados al proceso lógico y los datos de los peces que puedan influenciar

    La información que se envía es el identificador, la velocidad y la posición de los peces que pueden influenciar al nodo adyacente.

    Para la taxonomía de *master-only*, esta información se empaqueta en un bloque entero y no se aplica ninguna técnica de granularidad. Esta alternativa

de comunicaciones es la que se ha utilizado en los proyectos anteriores del departamento sobre el modelo del banco de peces *Huth and Wissel* [1,2].

En la siguiente figura se puede observar el diagrama de flujo de la implementación paralela híbrida *master-only.* El diagrama de flujo es igual que el del simulador MPI puro, mostrado en la figura 11, y así debe ser ya que la base del algoritmo paralelo y de la resolución del modelo biológico es la misma. No se observa ninguna bifurcación ni tarea diferente. Lo que aporta de diferente este simulador es que a ciertas tareas se les aplica paralelización a nivel de bucle mediante directivas OpenMP favoreciendo el acceso a la memoria compartida del procesador. El código que se ejecuta es el mismo en ambos simuladores, MPI puro y *master-only*, pero en éste último están activas las directivas OpenMP.

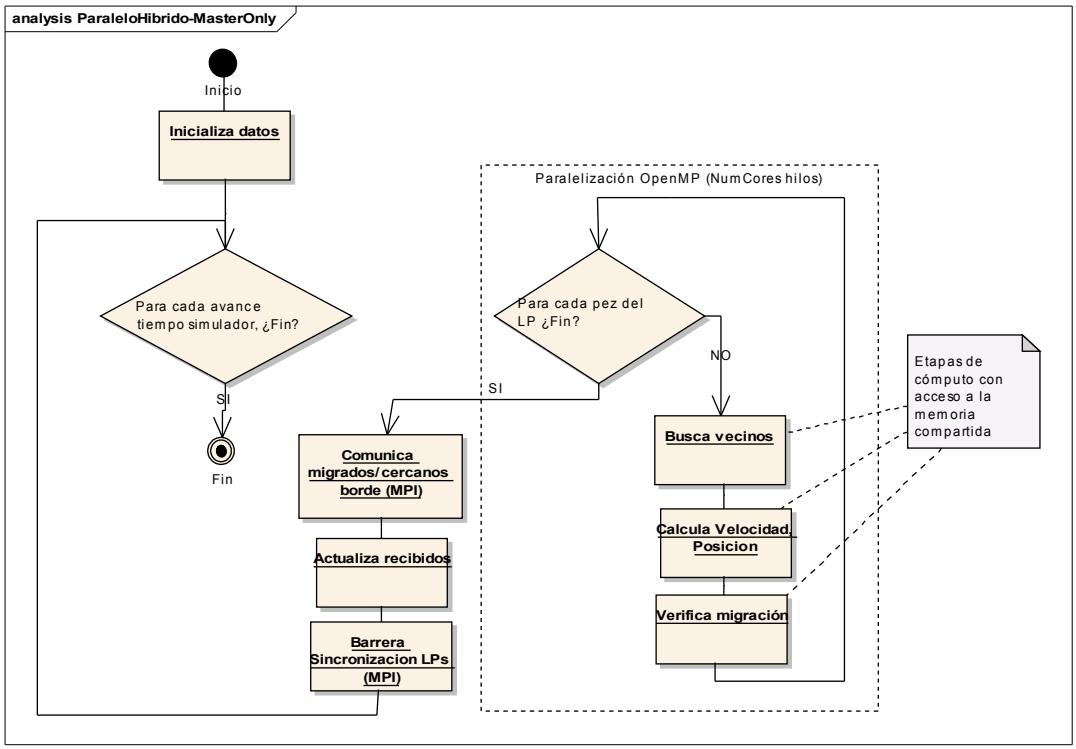

Figura 12: Diagrama flujo simulador paralelo híbrido *master-only*

Implementación paralela híbrida *solapamiento comunicaciones-cómputo*

Finalmente, se implementará el simulador con la programación híbrida *solapamiento comunicaciones-cómputo,* que es la que tiene mayor dificultad.

Esta implementación del simulador también presenta una primera etapa de inicialización de datos, en la que cada proceso lógico recupera, de un único fichero, los datos que le corresponden del banco de peces. Serán los peces que están en el intervalo del espacio de simulación y los de los procesos lógicos adyacentes que le pueden influenciar.

A partir de ese momento, hasta que se haya avanzado tantas veces en el tiempo, como tenga configuradas el simulador, cada proceso lógico realiza las siguientes tareas:

- Etapa de cómputo: Las iteraciones sobre los peces del proceso lógico se realizan en paralelo en todos los *cores* -1 del procesador del nodo mediante OpenMP. Por cada pez que gestiona el proceso lógico, se realizan las siguientes tareas:
    - Búsqueda de sus vecinos entre todos los peces del proceso lógico
    - Selección de los cuatro vecinos más cercanos, aplicando los criterios explicados en el apartado 3.2. En esta etapa, ya que es un bucle, también se aplica OpenMP
    - Cálculo de la nueva velocidad del pez como reacción a la influencia de los vecinos seleccionados, aplicando los criterios explicados en el apartado 3.2
    - Cálculo de la nueva posición según la nueva velocidad y la antigua posición, aplicando los criterios explicados en el apartado 3.2
- Etapa de comunicaciones. En paralelo al cómputo en el nodo. Hay un *core* con un hilo especializado en las comunicaciones. Mientras se van procesando todos los peces, se procede a realizar las operaciones de comunicaciones, todas mediante mensajería MPI:
    - Enviar a los procesos adyacentes los peces migrados y los datos de los peces influyentes, por bloques a medida que se vaya decidiendo su envío
    - Recibir datos de los procesos adyacentes sobre peces migrados al proceso lógico y los datos de los peces que puedan influenciar en el proceso lógico
    - Hay un flujo de estado para saber si ya se han acabado de enviar los datos de un hilo de cómputo: se van enviando en bloques hasta que se ha completado el cálculo de los peces de ese hilo y por último se envía un mensaje de finalización
    - Los envíos y las recepciones se hacen asíncronamente para facilitar el solapamiento de la comunicación con el nodo adyacente, ya sea tanto para recibir como para enviar.

  La información que se envía es el identificador, la velocidad y la posición de los peces que pueden influenciar al nodo adyacente.

Para la taxonomía de *solapamiento comunicaciones-cómputo*, las comunicaciones se realizan por etapas, independizando los envíos y recepciones de cada uno de los hilos del

nodo. Con esta estrategia, a medida que se calcula el modelo del conjunto de peces que corresponde a cada hilo, el hilo que está gestionando las comunicaciones se encargará de enviar los peces que hayan migrado y/o los que influyan en cada uno de los procesos adyacentes del proceso lógico actual. Se van enviando los datos dependientes en bloques hasta que se ha completado el cálculo de los peces de ese hilo y por último se envía un mensaje de finalización de ese hilo.

De esta forma, se consigue adaptar los envíos y recepciones a la velocidad de cómputo de los diferentes hilos, logrando que el solapamiento de las comunicaciones sea más efectivo, al no tener que esperar a que todos los hilos de ese proceso lógico hayan terminado su cómputo. El hilo especializado para las comunicaciones es el que está atento a que haya elementos a enviar de cualquiera de los hilos restantes del procesador.

En la siguiente figura se puede observar el diagrama de flujo de la implementación paralela híbrida *solapamiento comunicaciones-cómputo*. Como puede observarse, aquí ya hay un flujo diferente a los anteriores, ya que en cierto instante, se lanza el hilo de gestión de las comunicaciones que trabaja en paralelo a los hilos que computan el modelo.

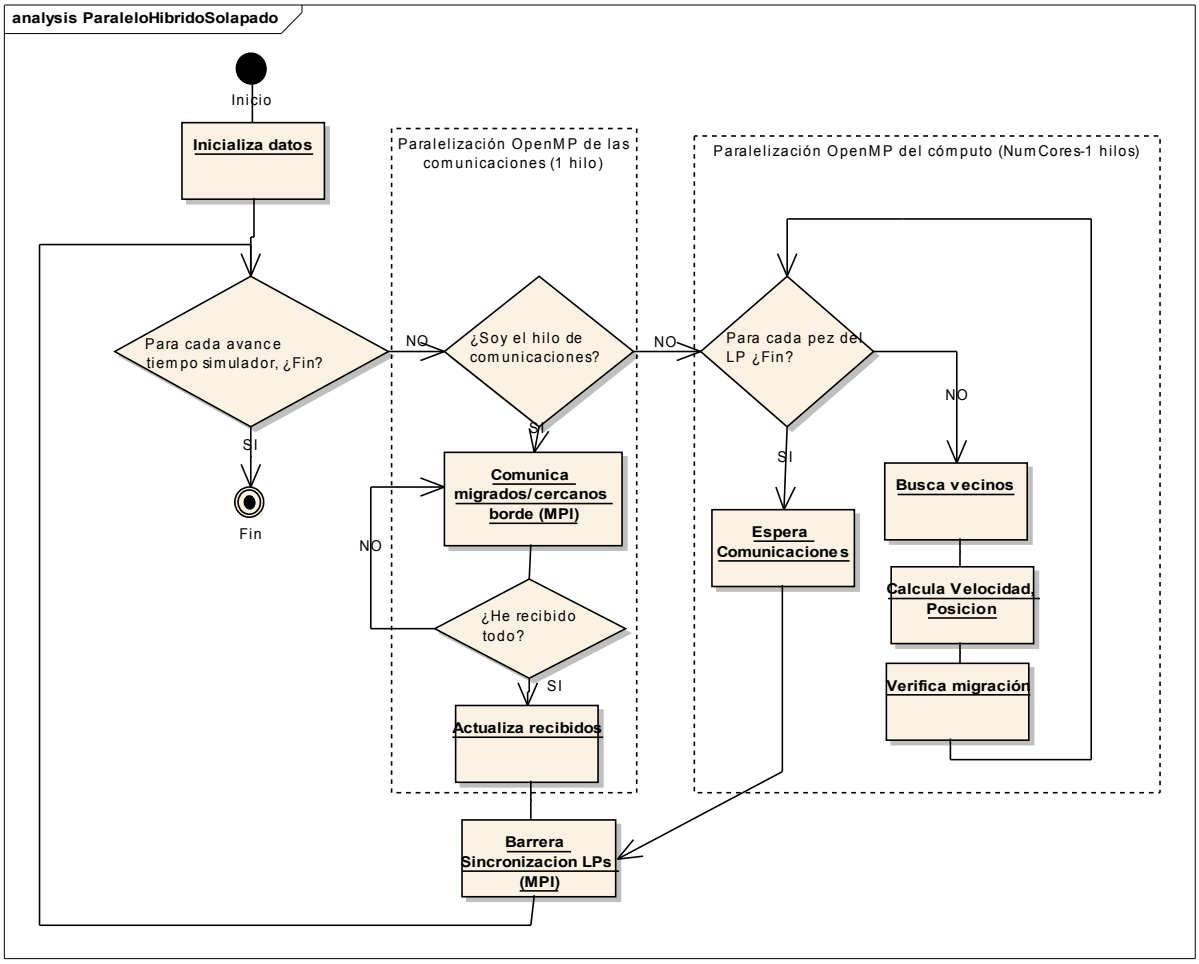

Figura 13: Diagrama flujo simulador paralelo híbrido *solapamiento comunicaciones-cómputo*

3.6 Estructuración del código

Se han clasificado el conjunto de clases del simulador en tres categorías:

- Las relativas a la gestión del simulador únicamente, que han sido diseñadas y creadas para este trabajo: Fsim, Environment, Communication
- Las que dan soporte al modelo *Huth and Wissel* [1,2] propiamente dicho, que se han modificado y revisado para este trabajo: Fish, Individual, Vector3D
- Un conjunto de utilidades específicas, diseñadas y creadas para este trabajo: DiscAccess, Support

En la figura 14 se ilustra la mencionada clasificación de las clases del simulador en categorías:

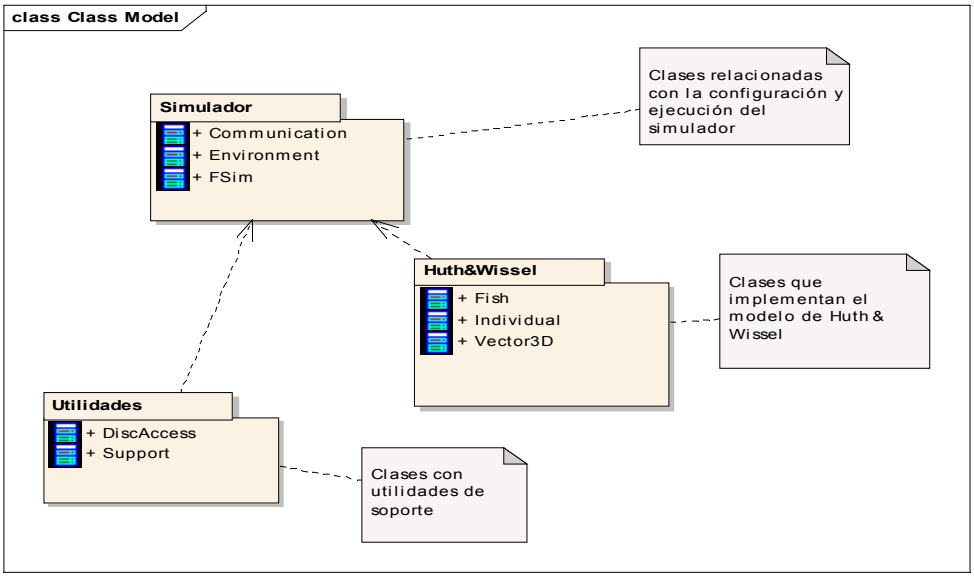

Figura 14: Categorías de clases

En la figura 15 se muestra el diagrama de clases y las dependencias entre ellas:

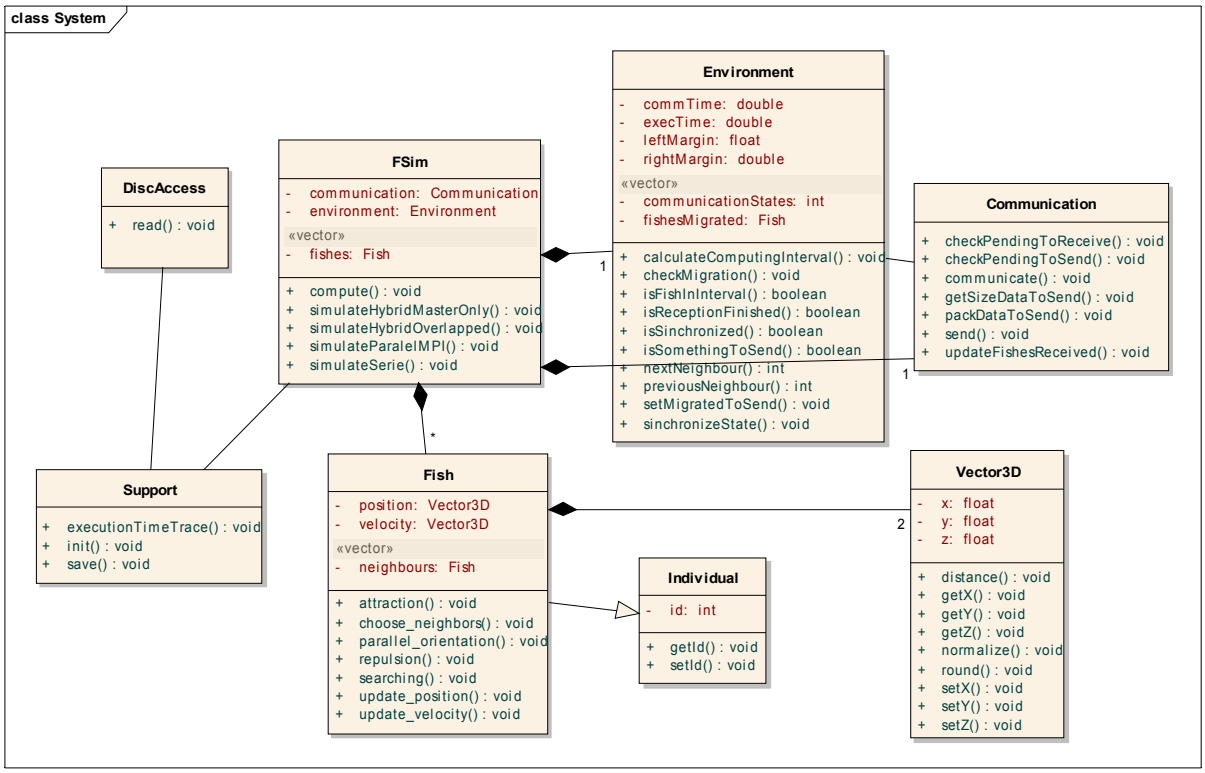

Figura 15: Diagrama de clases

Clase FSim

Se trata de la clase que contiene el punto de entrada al simulador. A partir de los parámetros de entrada es responsable de las siguientes acciones:

- Inicialización de las variables de entorno
- Leer la información del banco de peces a simular (posiciones y velocidades iniciales en el eje tridimensional)
- Invocar al simulador de la taxonomía híbrida que corresponda: serie, MPI pura, híbrida *master-only*, híbrida con *solapamiento comunicaciones-cómputo*
- Enviar eventos discretos para avanzar el simulador en el tiempo
- Mostrar los tiempos transcurridos, con una precisión de segundos. A saber: total de ejecución, tiempo en comunicaciones

Clase Environment

Se trata de la clase que guarda información del contexto de ejecución paralelo. Gracias a esa información el comportamiento del simulador se adapta dinámicamente a la arquitectura en la que se está realizando la ejecución. De esta forma, el simulador funciona transparentemente en cualquier arquitectura.

A continuación se enumera la información que permite conocer en tiempo de ejecución:

- En cuántos nodos se está ejecutando el simulador (se recupera de MPI)
- En qué nodo de ejecución me encuentro actualmente (se recupera de MPI)
- Posición de los nodos vecinos (se recupera de MPI)
- Cuántos hilos pueden ejecutarse en un nodo (se recupera de OpenMP)
- Estado actual de las comunicaciones con los nodos vecinos (depende de MPI)
- Cuál es la región del banco de peces que puedo tratar en mi nodo. Se trata de la sección transversal del eje de las X que se calcula dinámicamente a partir del resto de variables y de los peces que se encuentran en los extremos
- Adicionalmente, actúa como colector de datos de los tiempos de ejecución y comunicación

Clase Communication

Se trata de la clase que gestiona las comunicaciones con los procesos adyacentes. Esta clase es fundamental, ya que aporta el hecho diferencial de tener las comunicaciones solapadas o no, según estados internos que son asignados en tiempo de ejecución.

El modelo del comportamiento del banco de peces de *Huth and Wissel* [1,2] obliga a que cada pez tenga información actualizada de, al menos, los peces vecinos que se encuentren a cierta distancia de influencia. El peor caso es tomar en cuenta el radio de la atracción, que es el que se ha considerado en este proyecto.

Al dividir los datos según el eje de las X entre los nodos, deben realizarse comunicaciones con los nodos adyacentes (anterior y posterior) en los siguientes casos:

- Cuando un pez ha migrado. Sucede cuando al aplicar el movimiento al pez, éste se desplaza en el eje de las X fuera del intervalo que el proceso lógico gestiona.
- Cuando un pez puede influenciar a otro pez que se encuentre en un nodo adyacente. Esto sucede cuando un pez se halla muy cerca del extremo de la región que el proceso lógico gestiona. La información de este pez debe

comunicarse al proceso lógico adyacente por si tuviera peces cercanos a ese extremo que pudieran estar influenciados por los del proceso lógico adyacente.

Estas comunicaciones están basadas en MPI y se han encapsulado en esta clase para flexibilizar la llamada a la funcionalidad de comunicaciones desde cualquiera de las taxonomías híbridas de las que consta el simulador.

Los envíos y las recepciones se hacen asíncronamente para facilitar el solapamiento de la comunicación con el nodo adyacente, ya sea tanto para recibir como para enviar.

La información que se envía es el identificador, la velocidad y la posición de los peces que pueden influenciar al nodo adyacente.

Para la taxonomía de *master-only,* esta información se empaqueta en un bloque entero y no se aplica ninguna técnica de granularidad. Esta alternativa de comunicaciones es la que se ha utilizado en los proyectos anteriores del departamento sobre el modelo del banco de peces *Huth and Wissel* [1,2].

Para la taxonomía de *solapamiento comunicaciones-cómputo*, las comunicaciones se realizan por etapas, independizando los envíos y recepciones de cada uno de los hilos del nodo. Con esta estrategia, a medida que se va calculando el modelo del conjunto de peces que corresponde a cada hilo, el hilo que está gestionando las comunicaciones se encargará de enviar los peces que hayan migrado y/o los que influyan en cada uno de los procesos adyacentes del proceso lógico actual. Se van enviando en bloques hasta que se ha completado el cálculo de los peces de ese hilo y por último se envía un mensaje de finalización de ese hilo.

Clase Fish

Se trata de la clase que contiene las operaciones y propiedades que caracterizan a cada uno de los peces del banco de peces.

Como propiedades miembro, destacar la posición y la velocidad, que son del tipo Vector3d. Ambas están caracterizadas por tres variables de tipo *float* con la información de los ejes X, Y, Z. Además, al heredar de la clase Individual.cc, también tiene como propiedad un ID. Este identificador es único y es absolutamente necesario para poder realizar búsquedas con seguridad dentro del banco de peces.

Las principales operaciones del modelo de *Huth and Wissel* [1,2] son métodos miembros de esta clase. Operaciones como: repulsión, atracción, búsqueda de vecinos y actualización de velocidad forman parte de esta clase, con lo que se aplican a cada pez del banco de peces.

Clase Individual

Se trata de la clase que contiene la información identificadora del individuo. En este caso, un pez del banco de peces.

Clase VectoR3D

Se trata de la clase que contiene información de coordenadas en tres dimensiones con precisión *float*.

Además proporciona una serie de operaciones sobrecargadas para facilitar el manejo de la posición y velocidad del pez cuando tiene que aplicarse las reglas del modelo de *Huth and Wissel* [1,2].

Clase disc_access

Se trata de la clase que recupera de un fichero plano la información inicial del banco de peces.

En cada línea del fichero, se encuentra la información que caracteriza a un pez: identificador, posición (con las coordenadas X,Y,Z) y velocidad (con las coordenadas X,Y,Z).

Por cada línea leída, se instancia un nuevo pez, que se añade al vector que contiene el banco de peces y que es compartido por todo el simulador.

Clase Support

Se trata de la clase que contiene métodos para visualizar la información de ejecución e inicialización del Environment.

4 Experimentación y resultados

En este capítulo se expondrán los experimentos, su diseño y los resultados obtenidos para poder deducir cuál es la taxonomía híbrida más adecuada a utilizar según el tamaño del banco de peces datos a considerar con el modelo de *Huth and Wissel* [1,2].

El objetivo de la experimentación será estudiar el comportamiento del sistema frente a los cambios provocados en alguno de los factores variables del sistema. De esta forma, se

podrá discernir con claridad cuál de estos factores variables es el que influye en el sistema para conseguir un mejor rendimiento.

Se cuenta con un simulador que contiene tres implementaciones paralelas (MPI puro, híbida *master-only* e híbrida *solapamiento comunicaciones-cómputo* y se desea saber cuál de ellas tiene un mejor comportamiento en cuanto a rendimiento según el tamaño del banco de peces. El motivo por el que el tamaño del banco de peces es el factor considerado determinante, en este trabajo, es debido al hecho de que acceder a gran cantidad de datos en memoria (banco de peces grande) debería favorecer que las implementaciones híbridas ofreciesen un mejor rendimiento. Por este motivo, las bases que se asentarán primero son las de la verificación de que la implementación del simulador MPI pura escala en un sistema de altas prestaciones, que ya se había demostrado en anteriores trabajos, pero que en el presente servirá de punto de partida para comparar.

4.1 Infraestructura

En la tabla que se muestra a continuación se detallan las especificaciones del sistema *multicore* usado en la experimentación del presente trabajo.

Especificaciones sistema multicore (cluster Dell)
8 nodos Dell PowerEdge M600
Cada nodo con 2 procesadores Quad-Core Intel(R) Xeon(R) E5430 @2.66GHz
Cada procesador con 6MB de cache L2 y 16 GB RAM Fully Buffered DIMMs (FBD) 667MHz
SAS6ir (H/W based) RAID 1 120GB 2.5" SAS / Plain 2.5" SATA (7.2k rpm): 80GB
(Ethernet Pass-Through) Dual Gigabit Ethernet
OpenMPI 1.4.1 como librería MPI
GCC 4.4 como compilador
OpenMP 3.0 que viene habilitado en GCC mediante la opción -fopenmp

Tabla 1: Especificaciones sistema *multicore* usado en la experimentación

4.2 Diseño

En este apartado se detalla la estrategia del diseño de la experimentación, indicando de qué parámetros constantes consta el simulador, sus factores variables y dinámicos, cómo

se ha experimentado (instrumentación del código y configuración de la infraestructura de ejecución).

Los parámetros constantes son los relativos al modelo biológico y se desglosan en la tabla 2. El significado e influencia de estos parámetros ya ha sido descrito en los apartados relativos al modelo biológico.

Descripcion	**Valor**
Tamaño cuerpo pez (BL)	1
Radio repulsión	`0.8 * BL`
Radio atracción	`5.3 * BL`
Radio orientación paralela	`2.3 * BL`
Velocidad promedio	`0.1 * BL`
Ángulo muerto	`M_PI-M_PI/6.0`
Radio visión	`5.6 * BL`
Número vecinos que influencian	4
Espacio de simulación	2000x500x500
Iteraciones del simulador	50

Tabla 2: Parámetros constantes de la simulación

En la tabla 3 se encuentran los factores variables. Estos son: el número de procesos, sólo para el experimento de demostrar la escalabilidad del simulador MPI puro, y el tamaño del banco de peces. El tamaño del banco de peces es el factor considerado determinante, en este trabajo, ya que el hecho de tener que acceder a gran cantidad de datos en memoria (banco de peces grande) debería favorecer a que las implementaciones híbridas ofreciesen un mejor rendimiento. Por este motivo, el tamaño del banco de peces es el factor que se variará para observar el comportamiento del simulador (las tres implementaciones) en un sistema *multicore*.

Descripcion	**Valores**
Número de procesadores (sólo para demostrar la escalabilidad)	DE 1 A 8
Bancos de peces	Se han considerado los siguientes tamaños: 65000, 132000, 264000, 512000,1024000

Tabla 3: Factores variables

En la tabla 4 se encuentran los factores dinámicos. Solamente es uno, el número de hilos, que en tiempo de ejecución está disponible para los simuladores híbridos.

Descripcion	**Valores**
Número de hilos (aplicable en el simulador *solapamiento comunicaciones-cómputo*)	4

Tabla 4: Factores dinámicos

El simulador se ha instrumentado de forma tal que pueden conocerse los tiempos en las partes más significativas: al final de todo el proceso y el consumido en comunicaciones. Los tiempos se toman con la función MPI_Wtime de MPI, que devuelve el tiempo en segundos que ha pasado desde un momento arbitrario del pasado.

En la toma del tiempo serie también se ha usado esta función. De esta forma se mantiene y asegura la homogeneidad de la medida de tiempo, ya que la ejecución serie es la misma que el resto pero sólo con un procesador.

Por otro lado, se ha configurado la librería OpenMPI para hacer un uso más adecuado de la arquitectura subyacente. OpenMPI permite indicar el número de nodos, procesadores y cores por procesador y dirigir la ejecución de los diferentes procesos al elemento de la arquitectura que más interese: nodo, procesador, *core*.

Mediante esta opción, se han lanzado los experimentos del simulador paralelo MPI puro contra cada nodo independientemente y los experimentos de los simuladores híbridos a cada uno de los procesadores de cada uno de los nodos.

Los peces de los distintos bancos se encuentran distribuidos homogéneamente en el espacio del simulador. De esta forma se consigue que cada proceso lógico simule la misma cantidad de individuos de media, y por lo tanto el sistema se encuentra balanceado, inicialmente. A medida que avanza el simulador en el tiempo, los peces pueden migrar de proceso lógico y el sistema se desbalancea, pero no es objetivo de este trabajo resolver esa problemática.

4.3 Resultados

En cada iteración de la simulación, por cada pez, se debe realizar el cálculo de la distancia con los peces de su proceso lógico evaluando la reacción frente a las influencias de sus vecinos.

La primera fase de la experimentación tiene que ver con la verificación de que el simulador paralelo MPI puro logra una escalabilidad similar a la alcanzada en proyectos anteriores que trabajaban con el modelo de *Huth and Wissel* [1,2]. Los resultados se

pueden ver en la tabla 5 y observar en la figura 16.

La segunda fase de la experimentación tiene que ver con las implementaciones híbridas. En las tablas 6, 7 y 8 se muestran los resultados obtenidos y las figuras 17 y 18 muestran gráficamente estos resultados.

La experimentación con el simulador MPI puro se realizó con la metodología que se explica a continuación. Para un determinado espacio de simulación y cantidad de individuos 65000, se ejecutó la simulación con 1, 2, 4, 6 y 8 procesadores consecutivamente. Cabe destacar que se ha configurado openMPI para que los procesos lógicos se ejecutaran en nodos separados, así se consigue que la comunicación entre procesos lógicos se realice a través de la red de interconexión, como en los trabajos anteriores.

En la tabla 5, se muestran los resultados obtenidos de esta experimentación. En ellos se puede observar que, efectivamente, el tiempo total de ejecución del simulador se reduce a medida que se aumentan el número de recursos, es decir, procesadores, implicados en la resolución del problema. Por lo tanto, la implementación MPI puro del simulador escala.

Número procesos lógicos	**Tiempo total ejecución (seg)**
1	28974,79
2	4492,22
4	1101,7
6	757,71
8	517,13

Tabla 5: Resultados ejecución MPI puro

La medida que ayuda a comprender verificar cuánto de más rápido es el simulador paralelo que el algoritmo serie es el *speedup*. El *speedup* cumple con la siguiente fórmula: $S_p = T_{serie} / T_p$, donde p es el número de procesadores, T_{serie} el tiempo de ejecución del simulador serie (con un sólo procesador) y T_p el tiempo de ejecución del algoritmo paralelo en p procesadores. En la Figura 16 se muestra el *speedup* para la implementación MPI puro, calculado hasta 8 procesadores.

El *speedup* linear o ideal se obtiene cuando se cumple que $S_p = p$. Los sistemas, al escalar, suelen tener como límite superior el *speedup* lineal. Sin embargo, vemos en la figura 16 que hay super-linealidad. El motivo de este comportamiento es que se reduce la

complejidad del algoritmo al incrementarse la cantidad de procesadores, tal y como se demostró en el trabajo de Mostaccio [4].

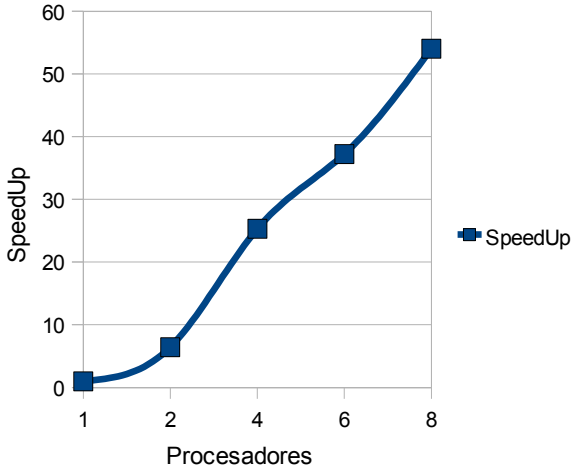

Figura 16: SpeedUp del simulador MPI puro

Ahora pasamos a la fase de experimentación con los simuladores híbridos. Como siguiente fase de experimentación se ha procedido a verificar el comportamiento de cada simulador, según el tamaño del problema, ya que el incluir la programación híbrida con memoria compartida debe aportar mejoras de rendimiento.

En este escenario de experimentación, el número de procesos lógicos es 8, se mantiene fijo, ya que se desea comparar los resultados de la experimentación con los obtenidos en la fase previa de experimentación con el simulador MPI puro.

En la tabla 6, se muestran los tiempos totales de ejecución de los tres simuladores: MPI puro, híbrido *master-only* y híbrido *solapamiento comunicaciones-cómputo*.

Tamaño muestra	**Tiempo total Master Only (seg)**	**Tiempo total solapamiento comunicaciones-cómputo (seg)**	**Tiempo total Puro MPI (seg)**
65000	485,75	517,73	517,13
132000	1947,48	2258,85	2132,83
264000	8000,65	9249,9	8847,9
512000	31980,55	37000,58	35166,05
1024000	130525,53	143101,95	140675,63

Tabla 6: Resultados tiempo total ejecución simuladores híbridos

En la tabla 7 se encuentran los tiempos de comunicaciones. Puede observarse que los tiempos dedicados a la comunicación son mucho menores, respecto al tiempo de cómputo total.

Tamaño muestra	**Tiempo comunicaciones Master Only (seg)**	**Tiempo comunicaciones solapamiento comunicaciones-cómputo (seg)**	**Tiempo comunicaciones Puro MPI (seg)**
65000	15,08	32,33	13,18
132000	52,08	51,98	46,38
262000	204,86	115,1	182,43
512000	509,66	680,3	530,9
1024000	1644,14	1858,15	1645,79

Tabla 7: Resultados tiempo total comunicaciones simuladores híbridos

En la tabla 8 se muestra el porcentaje que la comunicación representa en el tiempo total. Se muestra también con esta relación para mostrar con mayor claridad el impacto de las comunicaciones en el total del simulador. Este cálculo se ha obtenido mediante la siguiente relación:

$$Porcentaje_{comunicaciones} = T_{comunicaciones} * 100 / T_{total}$$.

Tamaño muestra	**Porcentaje tiempo comunicaciones (simulador master-only)**	**Porcentaje tiempo comunicaciones (simulador solapamiento comunicaciones-cómputo)**	**Porcentaje tiempo comunicaciones (simulador MPI puro)**
65000	3,10%	6,24%	2,55%
132000	2,67%	2,30%	2,17%
262000	2,56%	1,24%	2,06%
512000	1,59%	1,84%	1,51%
1024000	1,26%	1,30%	1,17%

Tabla 8: Porcentaje de tiempo que representan las comunicaciones

En la figura 17 se muestra gráficamente los datos del porcentaje de tiempo de comunicaciones de la tabla 8:

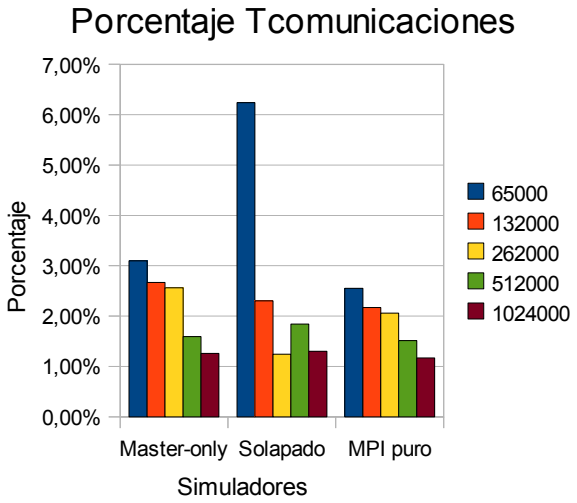

Figura 17: Porcentajes de tiempo de comunicaciones

Como previamente se ha sentado la base de que el simulador MPI puro tiene la escalabilidad esperada, se utilizará para demostrar cuánto se mejora con los simuladores híbridos, comparándolos con la implementación paralela MPI puro. Se utilizará la relación de porcentaje para mostrar la mejora obtenida, siguiendo la siguiente ecuación:

$$Porcentaje_{mejora} = (T_{PuroMPI} - T_{Híbrido}) * 100 / T_{PuroMPI}$$

En la Figura 18 se muestra el porcentaje de mejora de las implementaciones híbridas respecto la paralela MPI pura. Como puede observarse, sólo el simulador híbrido *master-only* mejora el rendimiento de la implementación MPI pura.

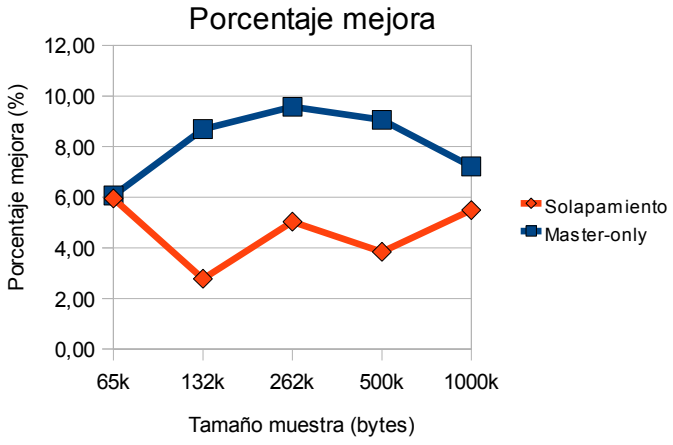

Figura 18: Porcentaje de mejora del tiempo de ejecución con los simuladores híbridos

Esto es debido a la propia definición del modelo, ya que cada proceso lógico trata con una gran cantidad de datos. El modelo del banco de peces *Huth and Wissel* [1,2] realiza iterativamente un cómputo relativamente sencillo, pero que debe tener en cuenta los peces que pertenezcan a una determinada región: dependencia con los datos.

Aunque hay gran cantidad de comunicaciones entre procesos lógicos, la cantidad de cómputo es más significativa en cuanto a tiempo de ejecución que el requerido para las comunicaciones.

A pesar de que se tenía expectativas de que la taxonomía híbrida *solapamiento cómputo-comunicaciones* ofreciera un mejor rendimiento que la *master-only*, no ha sido así. Hay varias explicaciones que avalan este hecho:

- Comunicaciones balanceadas intra-node versus extra-node: en realidad, debido a la arquitectura del entorno de experimentación, la mitad de las comunicaciones no sale del nodo: se comunican procesadores adyacentes que están en el mismo nodo.
- Se está dedicando un hilo de cuatro a comunicaciones. Resultan excesivos los recursos que se le dan a las comunicaciones (un 25%). El tiempo de comunicaciones es realmente muy pequeño en el caso de este modelo.

Cabe destacar que se tratan de nodos con dos procesadores *quad-core*. En cada procesador, se aplica el modelo a los peces que pertenezca a su región: se favorece la localidad de datos. A medida que crece el tamaño del banco de peces, como hay más migraciones de peces entre procesadores, o procesos lógicos, el tiempo requerido para las comunicaciones aumenta, pero no tanto como para perder peso en la mejora del paralelismo frente al compartir la memoria en el procesador por parte de todos los nodos (*master-only*).

Se puede observar también que la mejora del simulador de taxonomía *master-only* va aumentando un poco a medida que aumenta el tamaño del banco de peces hasta que llega a un punto de inflexión máximo y empieza a decrecer la mejora. Esto es debido a que hay ya tal cantidad de peces a compartir en la caché L2 que no caben en esta y en realidad los procesos de los procesadores tienen fallos de memoria y tienen que cargarla de nuevo.

Analizando el comportamiento de las comunicaciones, puede observarse en la tabla 7 que son tiempos muy pequeños en comparación con el tiempo de cómputo. Esta información refrenda el hecho de que la implementación híbrida *solapamiento comunicaciones-cómputo* no puede dar una mejora en el rendimiento mayor que la *master-only* ya que se pierde capacidad de cómputo (que en porcentaje es mucho más importante) a favor de solapamiento de comunicaciones (que no necesita tantos recursos).

5 Conclusiones y líneas futuras

Ha podido demostrarse, mediante la experimentación, la premisa de que la programación híbrida mejoraría el comportamiento del simulador *Huth and Wissel* [1,2], logrando independencia total de la arquitectura del sistema *multicore* subyacente, ya que mediante la configuración de la librería de ejecución puede lanzarse las ejecuciones a los activos de cómputo más adecuados. Se ha verificado mediante todos los experimentos realizados que ambas taxonomías híbridas mejoran el rendimiento del simulador paralelo MPI puro.

Como se ha avanzado en los primeros capítulos del trabajo, era necesaria la etapa de experimentación para poder afirmar con certeza cuál de las taxonomías híbridas proporciona mejor rendimiento para el modelo biológico de *Huth and Wissel* [1,2]. El volumen de cómputo utilizando datos compartidos es lo suficientemente representativo como para poder aplicar programación híbrida de taxonomía *master-only*. También se ha podido comprobar que este volumen de cómputo requiere de tanto tiempo de ejecución respecto a las comunicaciones que no se consigue mejora con la taxonomía híbrida *solapamiento comunicaciones-cómputo*. Sin embargo, es importante recordar que estos resultados han sido obtenidos en un escenario en el que como no hay estímulos externos y el banco de peces se comporta con un elevado grado de polarización y cohesión, se está favoreciendo el uso de variables en memoria ya que es poco probable que cambien los peces de proceso lógico.

A pesar de que se tenía expectativas de que la taxonomía híbrida de *solapamiento comunicaciones-cómputo* ofreciera un mejor rendimiento que la *master-only* no ha sido así. Hay varias explicaciones que avalan este resultado:

- Comunicaciones balanceadas intra-node versus extra-node: en realidad, debido a la arquitectura del entorno de experimentación, la mitad de las comunicaciones no sale del nodo: se comunican procesadores adyacentes que están en el mismo nodo.
- Se está dedicando un hilo de cuatro a comunicaciones. Resultan excesivos los recursos que se le dan a las comunicaciones. El tiempo de comunicaciones es realmente muy pequeño en este modelo biológico comparado con el tiempo de cómputo

Aunque se podría experimentar en sistemas *multicore* con más *cores*, el tiempo dedicado a las comunicaciones es tan pequeño en proporción al cómputo que puede deducirse que la taxonomía híbrida *solapamiento comunicaciones-cómputo* no mejoraría respecto a la *master-only*.

Adicionalmente destacar que el usar la infraestructura de ejecución correctamente configurada para el sistema *multicore* final ha permitido obtener unos resultados más ajustados a la realidad.

Como líneas futuras se podrían analizar aspectos más concretos de la arquitectura *multicore* subyacente como la asignación y planificación de hilos, configuración de la arquitectura subyacente y su red de interconexión, y cómo podría adaptarse esta topología al simulador del modelo biológico y sus datos.

La posibilidad de balancear hilos es otra alternativa para ayudar en la resolución de la problemática propia del modelo en el que al cabo de un tiempo la carga, peces, está desbalanceada.

6 Referencias

[1] Huth A., Wissel C., The simulation of the movement of fish schools, Journal of Theoretical Biology, 156(3): pp. 365–385, 1992.

[2] Huth A., Wissel C., The simulation of fish schools in comparison with experimental data, Ecological Modeling, 75: pp. 135–145+, 1994.

[3] Mostaccio D., Dalforno C., Suppi R., Luque E., Distributed Simulation of Large-Scale Individual Oriented Models, Journal of Computer Science & Technology, 6(2): pp. 59– 65, 2006.

[4] Mostaccio D., Simulación de Altas Prestaciones para Modelos Orientados al Individuo, PhD thesis, Computer Architecture and Operating Systems Departament, Universidad Autónoma de Barcelona, 2007.

[5] Dalforno C., Increasing the Scalability and the Speedup of a Fish School Distributed Simulator, MsC thesis, Computer Architecture and Operating Systems Departament, Universidad Autónoma de Barcelona, 2007.

[6] Jost G., Jin H., Mey D., Hatay F.F., Comparing the openmp, mpi and hybrid program-

ming paradigms on an SMP cluster, Proceedings of EWOMP, 2003.

[7] Mostaccio D., Suppi R., Simulación Distribuida de Modelo Orientados al Individuo utilizando MPI, CACIC 2004 - X Congreso Argentino de Ciencia de la Computación, La Matanza – Argentina, 2004.

[8] Mostaccio D., Suppi R., Simulation of Ecologic Systems Using MPI, EuroPVM/MPI 2005, Sorrento – Italia, 2005.

[9] MPI specification 2.2, http://www.mpi-forum.org/docs/, consultado Junio 2010.

[10] OpenMP specification 3.0, http://openmp.org/wp/openmp-specifications/, consultado Junio 2010.

[11] Solar R., Simulación de Altas Prestaciones sobre Modelos Complejos en Sistemas Orientados al Individuo, MsC thesis, Computer Architecture and Operating Systems Departament, Universidad Autónoma de Barcelona, 2009.

[12] Suppi R., Munt P., Luque E., Using PDES to Simulate Individual-Oriented Models in Ecology: a case study, ICCS 2002 – Proceedings of the International Conference on Computational Science-Part I – LNSC 2339 Amsterdam, The Netherlands: Springer –Verlag, pp. 107-116, 2002.

[13] Rabenseifner R., Hager G., Jost G., Hybrid MPI/OpenMP Parallel Programming on Clusters of multicore SMP Nodes, Proceedings of the 17th Euromicro International Conference on Parallel Distributed and Network-Based Processing (PDP 2009), Weimar, Germany, pp. 427–436, 2009.

[14] Sato T., Kitawaki S., Earth Simulator Running, ISC2002 - International Supercomputing Conference, Heidelberg, 2002.

[15] Adhianto L., Chapman B., Performance Modeling of Communication and Computation in Hybrid MPI and OpenMP Applications, 12th International Conference on Parallel and Distributed Systems - Volume 2 (ICPADS'06), pp. 3-8, 2006.

[16] Rabenseifner R., Hybrid Parallel Programming: Performance Problems and Chances, Proceedings of the 45th Cray User Group Conference, Ohio, May 12-16, 2003.

[17] Drosinos N., Koziris N., Performance Comparison of Pure MPI vs Hybrid MPI-Open-MP Parallelization Models on SMP Clusters, 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Papers, vol. 1, pp. 15a, 2004.

[18] Rabenseifner R., Hybrid parallel programming on HPC platforms, Proceedings of the Fifth European Workshop on OpenMP, EWOMP '03, Aachen, Germany, September 22-26, pp. 185–194, 2003.

[19] Top500 Supercomputing sites, http://www.top500.org/, consultado Junio 2010.

[20] Pitcher T.J., Magurran A.E., Edwards J.I., Schooling mackerel and herring choose neighbours of similar size, Mar Biol 86: pp. 319–322, 1985.

[21] Lorek H., Sonnenschein M., Using Parallel Computers To Simulate Individual-Oriented Models In Ecology: A Case Study, 1995 European Simulation Multiconference (ESM), 1995.

[22] 32 OpenMP traps for C++ developers, http://software.intel.com/en-us/articles/32-open-mp-traps-for-c-developers/, consultado Junio 2010.

[23] OpenMPI, http://www.open-mpi.org, consultado Junio 2010.