Mejoras en la eficiencia mediante *Hardware Locality* en la simulación distribuida de modelos orientados al individuo*

Silvana Lis Gallo [2, 3], Francisco Borges [1], Remo Suppi [1], Emilio Luque [1], Laura De Giusti [2], Marcelo Naiouf [2]

[1] Departamento de Arquitectura de Computadoras y Sistemas Operativos, Universitat Autònoma de Barcelona, Bellaterra, 08193, Barcelona, España.
[2] Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática, Universidad Nacional de La Plata, 50 y 120 2do piso, La Plata, Argentina.
[3] Becaria CONICET, Argentina.
{sgallo, ldgiusti, mnaiouf}@lidi.info.unlp.edu.ar
{Remo.Suppi, Emilio.Luque}@uab.cat, francisco.borges@caos.uab.cat

Resumen. La simulación de altas prestaciones aplicada a modelos orientados al individuo es de gran interés en la comunidad científica por la precisión que aportan sus datos, pero por contrapartida necesita grandes capacidades cómputo. Por ello, es necesaria la utilización de técnicas y métodos que permitan aprovechar toda la potencia de cómputo disponible para obtener el máximo *speedup* y eficiencia sobre la arquitectura. Por otro lado, el incremento del número de *cores*, cachés compartidas y memoria de los nodos ha introducido una complejidad en la arquitectura que puede afectar seriamente a las prestaciones/eficiencia de las aplicaciones si no se hace una distribución correcta teniendo en cuenta la jerarquía subyacente. El presente trabajo muestra las mejoras introducidas en la simulación distribuida de un modelo orientado al individuo (*Fishschools*) en sistemas *multicores* utilizando *hardware locality* (hwloc), la cual provee información sobre los procesadores de los nodos. Esta información será utilizada por la aplicación para adaptar las estrategias de ubicación de los procesos dependiendo de la afinidad hardware.

Palabras Clave: Simulación Paralela y Distribuida. Modelos Orientados a Individuos. Simulación de eventos discretos. Cluster de multicore. Evaluación de prestaciones.

1 Introducción

El estudio de la dinámica de poblaciones es un área de gran interés en el ámbito académico y ha sido el objetivo fundamental en el desarrollo de modelos biomatemáticos y una herramienta necesaria en la ecología demográfica para analizar y cuantificar las variaciones que sufren ciertas poblaciones a través del tiempo. Existen dos enfoques con los cuales se puede modelar la dinámica de poblaciones:

* El presente trabajo ha sido financiado por el proyecto del MICINN-España TIN2007-64974 y MINECO-España TIN2011-24384.

modelado basado en ecuaciones y modelado orientado a individuos. En los modelos basados en ecuaciones, las propiedades del sistema se obtienen mediante la resolución de un sistema de ecuaciones generalmente diferenciales y se obtienen resultados globales para el conjunto y con un alto grado de abstracción. En los modelos orientados a individuos, las propiedades del sistema emergen como resultado de la interacción y del comportamiento de los individuos del sistema y puede proporcionar resultados más cercanos a la realidad y con un alto grado de detalle. Estos modelos son de alta complejidad por lo cual no es posible su resolución analítica y deben hacerse por medio de simulación computacional.

Por otro lado, la aparición de arquitecturas distribuidas y procesadores con varios núcleos ha permitido el desarrollo de modelos complejos y a gran escala utilizando técnicas de simulación distribuida para reducir los tiempos de ejecución y obtener resultados en tiempo aceptables. No obstante, el aumento del número de procesadores interconectados a través de una red permite ejecutar gran cantidad de procesos, pero se debe tener especial cuidado en el impacto de las comunicaciones. Rápidamente se pueden degradar las prestaciones de la aplicación ya que el tiempo de comunicación puede ser más elevado que el cómputo paralelo y que en consecuencia no haya beneficios reales en su ejecución distribuida.

En este sentido, se han desarrollado técnicas y métodos que permiten simular modelos orientados al individuo de gran escala con el fin de disminuir el tiempo total de ejecución. La investigación realizada en [3] tiene por objetivo la simulación distribuida de *Fishschools*, un modelo orientado al individuo complejo, en el cual se demuestran las posibilidades de la simulación distribuida aplicando algoritmos de simulación conservativos. El trabajo realizado con este tipo de modelos tiene diferentes puntos que pueden comprometer las prestaciones y la escalabilidad del modelo: la asignación de individuos a los nodos de cómputo, cómo se agrupan para reducir la complejidad del algoritmo y el balanceo dinámico de carga o la importancia de cómo se realizan las comunicaciones entre los procesos distribuidos [5, 6, 16]. Todos los experimentos realizados demuestran que el simulador logra valores de *speedup* muy buenos para este tipo de modelos y que el modelo de simulación es escalable.

Sin embargo, ¿existe la posibilidad de mejorar la eficiencia del simulador y aprovechar de forma más adecuada la arquitectura subyacente?. Sería posible mejorando uno de los aspectos importantes y no considerados hasta el momento: la complejidad de la topología del hardware y la forma en que se realiza la distribución de los procesos sobre los *cores* de la arquitectura y cómo afectan las diferentes jerarquías de memoria a la ejecución de procesos. El objetivo del presente trabajo es analizar cómo el sistema operativo realiza la asignación de procesos a los nodos de cómputo-*cores* y cómo esto afecta a las comunicaciones (OpenMPI) entre los diferentes cores-nodos para obtener así la mejor estrategia con el fin de mejorar la eficiencia de la simulación bajo estudio utilizando *hardware locality* [18] (API que se describe en la sección 3 y cuya aplicación permite realizar las asignaciones de procesos de la simulación distribuida a los diferentes *cores*).

Este artículo se organiza con la siguiente estructura. En la sección 2 se presentan los modelos orientados a individuos, el simulador de banco de peces original y un resumen de las características anteriormente implementadas en el mismo. La sección 3 detalla las estrategias para utilizar la información de la arquitectura para la selección

y asignación de procesos a *cores* y en la sección 4, se describen las pruebas realizadas y resultados obtenidos en las mismas. Por último, en la sección 5, se presentan las conclusiones y trabajos futuros.

2 Modelos orientados al individuo

Los modelos orientados al individuo (IoM) permiten entender la dinámica del comportamiento de un sistema y consisten en una cantidad fija de individuos autónomos, para los cuales se definen reglas de interacción y atributos individuales que se mantienen a lo largo del tiempo. Además, incorporan un entorno donde ocurren las diferentes interacciones y un conjunto de parámetros que permite modelar estas interacciones y su movimiento en un espacio tridimensional.

Estos modelos también permiten incluir diferentes tipos de individuos dentro del mismo modelo con diferentes reglas de comportamiento y distintos valores de los atributos para modelar la interacción entre especies, su vinculación con el entorno, y también permite incluir diferentes mecanismos de aprendizaje, energía, obstáculos, roles (presa-depredador), etc. Algunos modelos orientados a individuos son también espacialmente explícitos, como es el caso de las simulaciones en que los individuos son asociados a una ubicación en el espacio geométrico y que también pueden mostrar patrones de movimiento (por ejemplo, los individuos pueden cambiar su posición relativa en un espacio geométrico). En la literatura existen diversos estudios de simulaciones espacialmente explícitas orientadas a individuos, como es el caso de aves [2,7], insectos [4,8,9], mamíferos [10] y peces [11-15].

2.1 Fishschools y la simulación distribuida

El presente trabajo utiliza como IoM a *Fishschools* que es un modelo biológico ampliamente validado y que representa el comportamiento de un conjunto de peces que es considerado como uno de los grupos sociales más frecuentes en el mundo animal [11,1]. Esta agrupación social muestra propiedades emergentes complejas, como por ejemplo: una fuerte cohesión de grupo (se mantiene la formación de grupo a través del tiempo), y un alto nivel de sincronización (los peces se mantienen nadando hacia la misma dirección y con la misma rapidez).

Para la simulación de este modelo se ha desarrollado un simulador distribuido que implementa algoritmos conservadores para realizar la sincronización entre procesos lógicos [3], por lo que éstos se bloquearán hasta que el procesamiento sea seguro. Para describir el comportamiento de los peces, el modelo considera que cada pez cambia su posición y orientación en pasos discretos de tiempo (time-driven) y los nuevos valores dependen de la posición y orientación de un número fijo de vecinos cercanos que estarán en función de la visión de pez. La influencia de los vecinos para un individuo en particular depende de su posición tiempo-espacio y la selección de la influencia de los vecinos está considerada en tres áreas de visión: atracción, repulsión y orientación paralela, como se indica en la Figura 1.

Dependiendo de la posición espacial y temporal de sus vecinos, el pez elige entre los tres patrones de comportamiento: **repulsión** - para evitar la colisión entre peces

del mismo grupo (cambiando la orientación del ángulo mínimo de rotación, logrando que la orientación del pez y la orientación de su vecino sean perpendiculares, Figura 2a), **orientación paralela** - el grupo se mueve en la misma dirección (haciendo coincidir la orientación del pez con la orientación de sus vecinos, Figura 2b), y **atracción** - para mantener la cohesión del grupo (dirigiendo su orientación hacia la posición de su vecino, la Figura 2c).

Figura 1. Áreas de visión

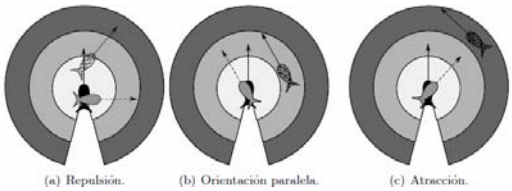

Figura 2. Patrones de comportamiento.

La orientación final del individuo surge a partir de la aplicación ponderada de las acciones resultantes con cada vecino y en el espacio tridimensional. Es importante notar que el modelo de simulación debe realizar este cálculo por cada individuo y para cada paso de simulación lo cual se deriva en altas necesidades de cómputo cuando se trabajan con un número importante de individuos (de 64k a 512k individuos en el caso tratado).

Con el fin de mejorar el rendimiento en la simulación, los autores muestran en [3] una importante contribución a la generalidad del modelo original -en dos dimensiones- [1] añadiendo la interacción depredador-presa, evasión de obstáculos cilíndrica, y una representación interactiva de los límites del mundo en el modelo 3D. Este modelo si bien es más complejo también produce resultados más cercanos a la realidad reduciendo el grado de abstracción y proporcionando buena escalabilidad y rendimiento cerca de los valores de *speedup* ideales.

En [5] se presenta un enfoque de agrupación en la distribución de los individuos en la arquitectura con el fin de obtener el mejor rendimiento de la simulación ya que el desbalance de carga es un problema importante en la simulación distribuida. El enfoque basado en clústeres consiste en determinar el conjunto de individuos óptimo a cada elemento de cómputo de forma tal que permita reducir la interacción entre los individuos que están lejos. En [6] considerando el problema de la distribución se presenta una nueva estrategia que complementa a la anterior y que incluye el equilibrio de carga dinámico para evitar que después de una asignación inicial óptima en los nodos de cómputo se tengan desbalances debido al movimiento de los individuos y el paso del tiempo de simulación. Esta estrategia de balanceo de carga se basa en reconfigurar y redistribuir la carga de trabajo local haciendo nuevas agrupaciones y migrando los individuos desde un nodo de cómputo hacia otro.

Finalmente, dado que la comunicación es uno de los puntos débiles de la simulación distribuida en [17], se comparan tres estrategias de comunicación implementadas en el simulador distribuido: comunicación asíncrona y síncrona de paso de mensajes y *bulk-synchronous parallel*. En este trabajo se demuestra que las simulaciones *time-driven* no siempre aumentan el rendimiento mediante el uso de estrategias de comunicación sincrónica. Los resultados demuestran que la comunicación sincrónica obtiene peores resultados en términos de tiempos de ejecución en comparación con la estrategia de comunicación asincrónica.

Una vez optimizados los aspectos del modelo, asignación y balanceo de carga de los individuos, y prestaciones de las comunicaciones queda por analizar y optimizar la eficiencia y uso de los recursos de la arquitectura, Es por ello que en el presente trabajo se estudia el impacto de la afinidad de procesos dinámica utilizando *Hardware Locality* [18] en el simulador distribuido de modelos orientados al individuo con el objetivo de analizar cuándo existen mejoras en las prestaciones y en la eficiencia en las arquitecturas de cómputo actuales.

3 *Hardware Locality* aplicada a la simulación distribuida de *Fishschool*.

Los procesadores *multicores* son ampliamente utilizados en la computación de altas prestaciones y es una tendencia donde la arquitectura es cada vez de mayor complejidad y donde adquiere especial relevancia el acceso no uniforme a la memoria de los nodos de cómputo. Es por ello que se requiere una asignación ordenada y en base a estrategias predefinidas de procesos y datos en función de su afinidad si se desea aprovechar al máximo las posibilidades de la arquitectura subyacente. Por otro lado hay que tener en cuenta que en grandes infraestructuras, generalmente controladas por sistemas de colas, las políticas de asignación de las colas pueden ser totalmente contrarias a las necesidades de la aplicación distribuida y el usuario debe disponer de herramientas que bajo unas situaciones predefinidas pueda asignar sus procesos de la forma más conveniente (y en forma dinámica) para la ejecución de la aplicación.

También es necesario tener en cuenta que la utilización de hardware de red puede resultar de acceso no uniforme ya que las tarjetas de interfaz pueden estar más cerca de algunos procesadores que de otros. Es por ello que se debe analizar estos aspectos pues pueden tener un impacto considerable en el rendimiento de las comunicaciones y afectar a las aplicaciones de paso de mensajes, como por ejemplo las que utilizan OpenMPI.

Una posible estrategia de trabajo es actuar sobre la colocación (*mapping*) de los procesos orientados a cómputo y de los orientados a comunicaciones para que de esta forma se pueda mejorar las prestaciones de la aplicación facilitando tanto el acceso a datos como el acceso a las interfaces de red para aquellos procesos que lo necesiten de forma intensiva.

Con el fin de poder explotar el potencial de las arquitecturas *multicores* para el simulador distribuido bajo estudio se ha utilizado la API hwloc [18] que permite realizar una abstracción de la topología hardware al desarrollador de la aplicación. Asimismo, estas características permiten en tiempo de ejecución realizar la asignación controlada de recursos (*cores*) a los procesos lógicos de la aplicación de acuerdo a la estrategia preseleccionada y el *hardware* subyacente donde se ejecutarán dichos procesos.

Normalmente, será el sistema operativo quien elegirá los núcleos donde se crearán los procesos MPI y estos podrán ser desplazados, de acuerdo a sus políticas de optimización, de un núcleo hacia otro como se muestra en la Figura 3 sobre 4 procesos MPI y su asignación a un procesador de 8 *cores* y dos *sockets*.

Sin embargo, esto puede limitar el rendimiento paralelo de la aplicación en el contexto de HPC ya que se pierden las ventajas de la memoria caché y la localidad cuando el proceso migra de un núcleo a otro. Como se puede observar, los procesos se han creado bajo un criterio predefinido por el SO con el que no compartirán la cache L2 y por lo cual no podrán compartir información entre los procesos. En cambio si se mantiene un proceso "unido" a un *core* de un núcleo y se evita el cambio de contexto tendrá como resultado una mejor utilización de caché y producirá una mejora de las comunicaciones entre procesos [16].

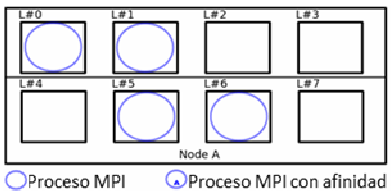

Figura 3. Asignación por el SO de cuatros procesos MPI

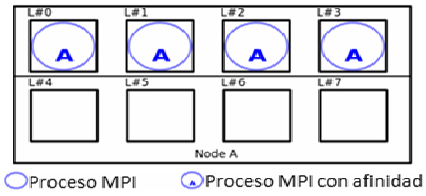

Figura 4. Cuatro procesos MPI asignado a cuatro cores por Hwloc

Para evitar estos problemas, cada proceso de MPI debería se asignado a un núcleo de forma consecutiva, de acuerdo con el número del MPI Rank. De esta forma, los procesos adyacentes estarán espacialmente cercanos y podrán tomar ventaja de L1 y L2 (Figura 4).

La estructura de datos utilizada en el simulador distribuido se basa en una lista de radio fijo de clústeres [5] (Figura 5), que se mantiene en cada proceso lógico de simulación (LP) y donde el proceso dispone de toda la información para realizar el siguiente paso de simulación. Es importante analizar que la forma de acceso a los datos y su asignación a elementos de cómputo puede favorecer las prestaciones teniendo en cuenta cómo se almacenan los individuos en un nodo y la forma en que se defina la afinidad.

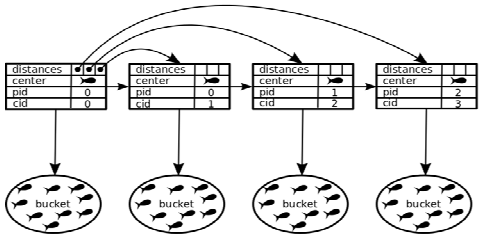

Figura 5. Estructura de datos que mantiene cada LP.

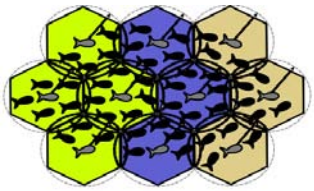

Figura 6. Áreas de clústeres adyacentes (mismo color) deberían ser asignadas a *cores* adyacentes.

Esta estructura de datos permite definir áreas en las que los individuos pueden interactuar sólo con los que pertenecen a las zonas adyacentes y ello permite reducir el tiempo de cálculo de los individuos que participan en el mecanismo de selección vecinos (Figura 6). Si estas áreas son asignadas a elementos que puedan compartir la caché del *socket* los procedimientos que utilizan datos de su área y de áreas vecinas (método de descubrimiento de vecinos) tendrán acceso a los datos de forma más eficiente que si se asignan a *cores* que no compartan esta memoria.

Para realizar la experimentación y analizar el impacto de la afinidad en procesos MPI, el código del simulador ha sido modificado para que permita obtener

conclusiones de cómo afecta la afinidad a la simulación distribuida de modelos orientados al individuo sobre una arquitectura *multicores*.

4 Pruebas realizadas

Para la experimentación se ha utilizado una arquitectura de tipo Blade de 12 bahías y donde cada *blade* dispone de 2 procesadores quad core Intel Xeón (e5405@2.0GHz) con una caché L1 privada de 64Kb (dividida en 32Kb para instrucciones y otros 32Kb para datos), y cache L2 de 2 x 6Mb entre par de núcleos. En relación a la memoria principal (RAM), 8 de los *blades* disponen de 10Gb compartida entre ambos procesadores y los 4 nodos restantes tienen 2 Gb en la misma configuración. La Figura 7 muestra la imagen generada por el comando *lstopo* de hwlock1.6.1.

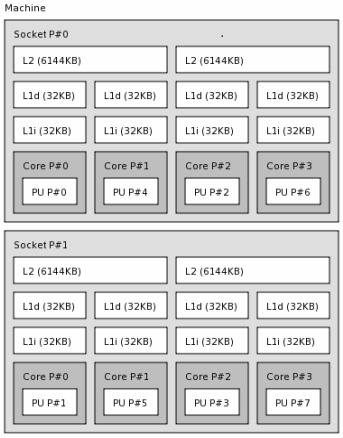

Figura 7. Arquitectura del *blade* obtenida por el comando *lstopo* (hwloc).

Para analizar el comportamiento de cada solución paralela implementada, se midió tiempo de cómputo y comunicaciones así como también se calcularon los valores *speedup*, y eficiencia para cada escenario. Por un lado, se realizaron pruebas con hasta 64 *cores* de la arquitectura en nodos con igual cantidad de memoria (10Gb). Los distintos escenarios de prueba se realizaron con la misma cantidad de ciclos de simulación (250 pasos), y escalando el tamaño de la población de individuos (8K, 16K, 32K, 64K, 128K individuos), y las diferentes cantidades de *cores* utilizados (en total 4, 8, 16, 32 y 64). Por otro lado, para analizar el impacto de la heterogeneidad en la solución, se realizaron pruebas con 64 y 96 nodos activando y desactivando el uso de *hwloc*.

4.1 Pruebas en el entorno homogéneo

Los valores obtenidos para los tiempos de cómputo y comunicación para cada uno de los escenarios son los visualizados en la Figura 8. Como se puede observar, en cuanto a los tiempos de cómputo (Figura 8a), el uso de *hwloc* logra una disminución para todos los escenarios de prueba. Por otro lado, cuando el escenario de prueba es grande (128K individuos), el tiempo de comunicación (Figura 8b), presenta una mayor

reducción que en los demás escenarios de prueba. Esto se debe a que la ejecución de la simulación para esta cantidad de *cores* implica una gran cantidad de mensajes, por lo que permite que el uso de la API *hwloc* sea más notable.

En la Figura 9, se muestran *speedup* y eficiencia obtenidos en las simulaciones. En todos los casos, se observa una mejora que alcanza 2%, siendo el mejor resultado el obtenido por el escenario 4c-8k, lo cual es esperable, porque cuando se incrementa el número de procesos/*cores*, la eficiencia disminuye ya que los mensajes y el número de procesos hace que los tiempos de espera por comunicación adquieran relevancia. Es importante notar que la distribución de los procesos (manual a través *hostfile*) no es la peor (ni aleatoria como podría ser en un sistemas de colas) ya que el algoritmo de asignación de sistema operativo está haciendo una asignación similar a la de hwloc.

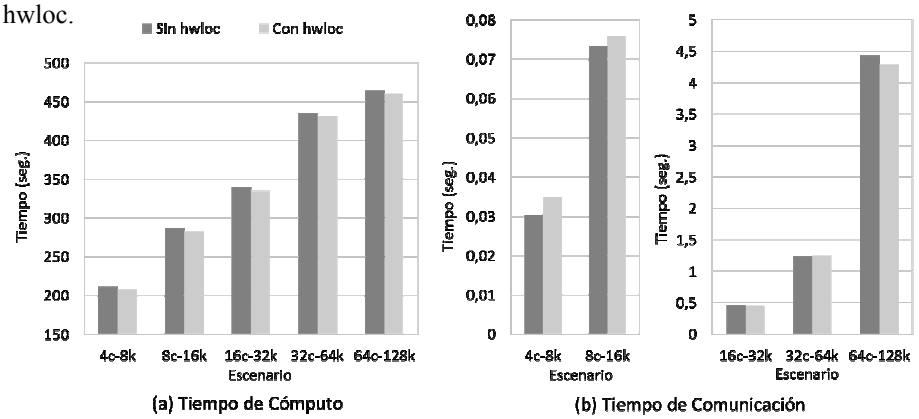

Figura 8. Tiempos de cómputo y comunicaciones para los diferentes escenarios.

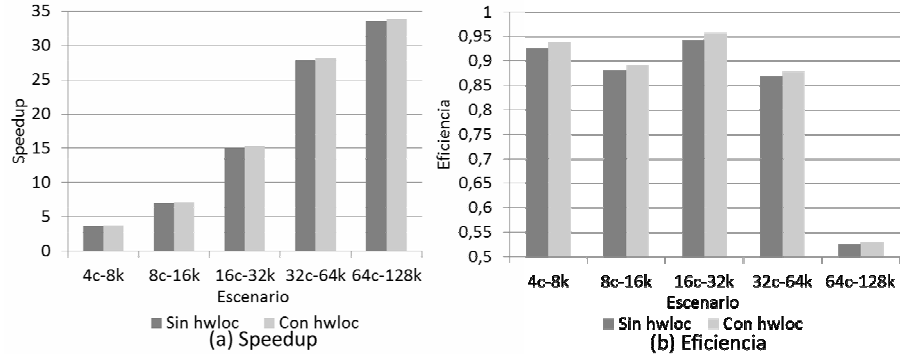

Figura 9. Speedup y eficiencia para las pruebas homogéneas.

Además, se debe considerar que las herramientas como OpenMPI se encuentran optimizadas para ambientes *multicore* (haciendo intercambios de punteros entre buffers de mensajes), por lo que, definir procesos en la misma y asignarlos manualmente (mediante un *hostfile)* permite realizar una buena distribución, que se ha visto mejorada con el uso de *hwloc*. Este resultado permite pensar que la incorporación de herramientas de programación de memoria compartida (como por ejemplo OpenMP) podría resultar de gran ayuda para el perfeccionamiento de las soluciones sobre clúster de *multicore*.

4.2 Pruebas en el entorno heterogéneo

En este caso se utilizaron las hojas de 2Gb y 10Gb intercaladas para la ejecución, con un escenario de 128K individuos, con 64 y 96 cores, permitiendo obtener los valores de eficiencia de la Figura 10. Como se observa, si el mismo caso de prueba se ejecuta con y sin *hwloc* en los distintos entornos (Figura 10a), se consigue mejor eficiencia en una arquitectura homogénea. Por otro lado, cuando se utiliza *hwloc* la eficiencia es levemente mayor, lo cual indica que esta solución es favorable en ambos entornos, y mayormente en el entorno heterogéneo. Al ser los incrementos de la eficiencia muy pequeños, se refuerza la necesidad de buscar una alternativa que optimice el trabajo dentro de cada nodo de acuerdo a su arquitectura (en especial L1 y L2).

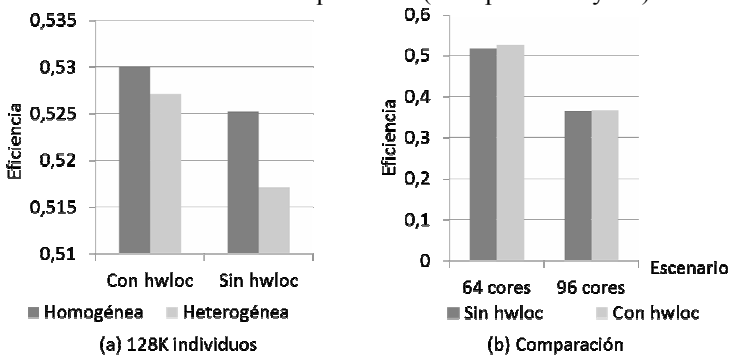

Figura 10. Eficiencia en soluciones con heterogeneidad de arquitectura

Por último, si se compara la eficiencia obtenida para un mismo caso de prueba (128K individuos), para un mayor número de procesadores en un ambiente heterogéneo (como es el caso de 96 *cores* en la Figura 10b), se puede observar que el uso de *hwloc* sigue siendo levemente favorable para la ejecución en el entorno heterogéneo.

5 Conclusiones y trabajos futuros

Como resultado del presente trabajo se obtuvo una reducida mejora que, a medida que el número de *cores* es más elevado, tiende a disminuir por el propio efecto de los mensajes. Además, considerando que la ejecución de las pruebas fue en un entorno sin manejo de colas, el SO realiza la asignación de manera similar a *hwloc*. Así mismo es importante notar que el simulador mantiene su escalabilidad y hwloc no perjudica a valores elevados de *cores*/procesos. En el sentido de los incrementos de prestaciones será necesario explorar nuevas alternativas que permitan reducir la cantidad de mensajes de la aplicación, como puede ser el uso de herramientas de programación sobre memoria compartida (por ejemplo OpenMP). Por otro lado, si bien el simulador hace un balance de carga, no lleva a cabo un balance de cómputo, que perjudica los resultados obtenidos, convirtiendo este aspecto en otro ítem a optimizar.

Agradecimientos

Los autores desean agradecer al Dr. Roberto Solar las sugerencias al presente trabajo así como las aportaciones e investigaciones realizadas en las versiones anteriores del simulador distribuido. El presente trabajo ha sido financiado por el proyecto MICINN-España TIN2007-64974 y proyecto MINECO-España TIN2011-24384.

Referencias

1. A. Huth, C.Wissel, The simulation of fish schools in comparison with experimental data, Ecological Modelling, State-of-the-Art in Ecological Modelling, Proceedings of ISEM's 8th International Conference.75-76 (1994) 135–146.
2. C. W. Reynolds, Flocks, herds and schools: A distributed behavioral model, SIGGRAPH Computer Graphics. 21 (1987) 25–34.
3. R. Solar, R. Suppi, E. Luque, High performance individual-oriented simulation using complex models, Procedia Computer Science 1 (1) (2010) 447 – 456.
4. X. Hu, Y. Sun, Agent-based modeling and simulation of wildland fire suppression, in: Proceedings of the 39th conference on Winter simulation, IEEE Press, USA, (2007) 1275–1283.
5. R. Solar, R. Suppi, E. Luque, High performance distributed cluster-based individual-oriented fish school simulation., Procedia CS 4 (2011) 76–85.
6. R. Solar, R. Suppi, E. Luque, Proximity load balancing for distributed cluster-based individual-oriented fish school simulations, Procedia Computer Science 9 (0) (2012) 328 – 337.
7. R. O. Saber, R. M. Murray, Flocking with obstacle avoidance: cooperation with limited communication in mobile networks, Proceedings 42nd IEEE Conference on Decision and Control, 2003, Vol. 2, (2003) 2022–2028.
8. E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm intelligence: from natural to artificial systems, Oxford University Press, USA, (1999).
9. J. Kennedy, R. C. Eberhart, Swarm intelligence, Morgan Kaufmann Publishers, USA, 2001.
10. S. Gueron, S. Levin, D. Rubenstein, The dynamics of herds: From individuals to aggregations, Journal of Theoretical Biology 182 (1996) 85–98.
11. A. Huth, C. Wissel, The simulation of the movement of fish schools, Journal of Theoretical Biology 156 (3) (1992) 365 – 385.
12. I. Aoki, A simulation study on the schooling mechanism in fish, Journal of the Japanese Society of Scientific Fisheries 48 (8) (1982) 1081–1088.
13. R. Vabø, G. Skaret, Emerging school structures and collective dynamics in spawning herring: A simulation study, Ecological Modelling 214 (2-4) (2008) 125–140.
14. J. K. Parrish, S. V. Viscido, D. Grnbaum, Self-organized fish schools: An examination of emergent properties, Biological Bulletin 202 (2002) 296–305.
15. J. C. Gonzlez, C. Dalforno, R. Suppi, E. Luque, A fuzzy logic fish school model, Lecture Notes in Computer Science, Vol. 5544 (2009) 13–22.
16. Open MPI Team. FAQ: General run-time tuning., Feb 2012. Visited on July 23, 2013.
17. R. Solar, F. Borges, R. Suppi, and E. Luque. Improving Communication Patterns for Distributed Cluster-based Individual-oriented Fish School Simulations. Procedia Computer Science, 18(0), (2013) 702-711.
18. François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, IEEE Computer Society Press, (2010) 180-186. DOI: 10.1109/PDP.2010.67.