Generación Automática de Funciones de Migración de Agentes en FLAME

Claudio Márquez[1], Eduardo Cesar[1], Joan Sorribes[1]

Resumen— **Las aplicaciones basadas en agentes son usadas para hacer grandes simulaciones de modelos de sistemas complejos. Los entornos HPC proporciona la capacidad de cómputo para simular problemas de mayor complejidad en donde se requieren agentes con reglas de interacción más complejas y también una mayor cantidad de agentes. En este documento se presenta una modificación del framework FLAME (Flexible Large-scale Agent Modeling Environment) para simulaciones distribuidas basadas en agentes que permite la generación automática de funciones para la migración dinámica de agentes entre las diferentes unidades de cómputo.**

Palabras clave— **Simulación Basada en Agentes, FLAME, Balanceo de carga, Sintonización.**

I. Introducción

Debido a las características que presentan las simulaciones del área ABMS (Agent-Based Modeling and Simulations), estas se podrían beneficiar por los sistemas HPC (High Performance Computing). Los entornos HPC permiten la ejecución de ABMS con muchos más agentes y reglas más complejas soportadas que en un entorno no distribuido, por lo que se pueden realizar experimentos con modelos más complejos y más realistas[1].

Estas simulaciones presentan variabilidad en la cantidad cómputo y de comunicación, debido a los altos niveles de interacción entre agentes y a las diferentes reglas de comportamiento que presentan muchos de estos modelos, por lo tanto se generan problemas de desbalanceo de carga durante el proceso de simulación.

La figura 1 muestra como en dos iteraciones algunos nodos poseen una mayor cantidad de agentes y en cambio otros muy pocos lo que provoca problemas de desbalance que se reflejaran en el tiempo necesario para la ejecución de la simulación.

Para resolver este tipo de problemas las plataformas de simulación deberían estar dotadas de mecanismos que permitan realizar migración de agentes entre las diferentes unidades de cómputo que se posea.

Actualmente se pueden encontrar pocas aplicaciones paralelas orientadas a entornos HPC, entre estas encontramos: Ecolab, Repast HPC y FLAME. Ecolab[2] es un entorno orientado a objetos escrito en C++ y MPI, esta herramienta presenta problemas de escalabilidad [3], en 2008 aparecieron las últimas publicaciones de la herramienta y el 2010 la última actualización, por lo que no hay más información sobre la continuidad de su desarrollo. Repast HPC[4] fue liberada el 5 de marzo del 2012, escrita en C++ y MPI para las operaciones paralelas. A diferencia de Ecolab, Repast HPC ha sido creada directamente para plataformas de cómputo distribuido a gran escala. Si bien ambas describen la necesidad migrar agentes e incluyen algún tipo de rutinas orientadas a la migración, no se automatizan de forma genérica rutinas que faciliten la migración por lo que el desarrollador debe implementar todo el proceso de migración. Finalmente está el framework FLAME[5] que es un generador de código fuente de simulación desarrollado desde el año 2006 hasta el día de hoy y se comienza a usar formalmente para modelado financiero en el proyecto EURACE[6]. FLAME está escrito en C usando MPI y puede ser usado en dominios tales como: economía, medicina, biología y en ciencias sociales.

El código generado por FLAME carece de la rutinas necesarias para la migración de agentes, por lo cual si se desea migrar agentes, estas rutinas deben ser implementadas directamente sobre el código generado.

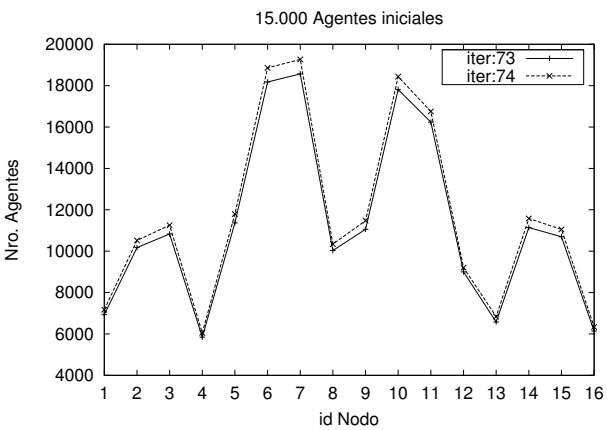

Fig. 1. Ejemplo del problemas de desbalance de carga en dos iteraciones (modelo epidemico SIR).

En este trabajo se presenta una modificación al framework FLAME que permite la creación en forma automática de las funciones necesarias para realizar migración dinámica de agentes entre procesos, esta mejora permitirá en el futuro poder tomar decisiones automáticas de sintonización en cuanto a la carga de trabajo. La estructura del trabajo incluye la descripción del funcionamiento de FLAME(II), posteriormente se discute la contribución propuesta(III) para más tarde presentar los casos de uso(IV). Finalmente las conclusiones(V) y trabajos futuros(VI).

[1]Dpto. de Arquitectura de Computadores y Sistemas Operativos, Universidad Autónoma de Barcelona, e-mail: claudio.marquez@caos.uab.es. eduardo.cesar@uab.cat. joan.sorribes@uab.cat.

II. FLAME

FLAME es desarrollado en la Universidad de Sheffield en colaboración con el Science and Technology Facilities Council (STFC) en el Reino Unido. FLAME puede ser usado en dominios de economía, medicina, biología y en ciencias sociales. Permite escribir diferentes modelos basados en agentes sobre un entorno de simulación común y posteriormente hacer simulaciones sobre diferentes arquitecturas paralelas (HPC) y GPUs de una manera sencilla.

A. Funcionamiento

Por si mismo FLAME no es un simulador sino es una herramienta capaz de generar el código fuente necesario para la simulación. A través de la definición del modelo y una serie de plantillas, FLAME genera automáticamente el programa de simulación en lenguaje C. La definición del modelo debe estar contenida en un fichero de formato XML más un fichero C que contiene las funciones asociadas a los agentes. En la figura 2 se muestra al lado izquierdo los ficheros del modelador requeridos por parte de FLAME para generar el código de simulación.

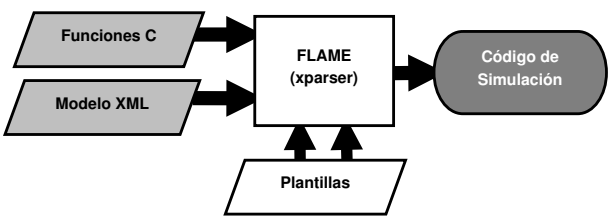

Fig. 2. Generación de codigo se simulación a través de FLAME.

El funcionamiento de FLAME está basado en máquinas de estados finitos conocidas como X-machines, las cuales contienen un número finito de estados, transiciones entre los estados y acciones. Para realizar la simulación, FLAME almacena cada agente como una estructura de datos X-machine, las cuales, a través de una serie de funciones de transición van modificando su estado y realizan los intercambios de mensajes necesarios entre agentes. Finalmente, el entorno de simulación tendrá un conjunto de X-machines compuestas por una serie de estados y una serie de funciones de transiciones entre estados.

Para realizar estas transiciones entre estados de un tipo de agente específico, FLAME crea una lista doblemente enlazada por cada estado de los agentes. Durante la simulación, para comenzar a realizar iteraciones sucesivas, las X-machine se insertan en la lista correspondiente al estado inicial y a cada X-machine se aplica la función de transición para luego ser insertadas en la lista del siguiente estado, este proceso se repite hasta llegar al estado final con lo que termina una iteración.

B. Paralelización

Las comunicaciones en entornos HPC se realizan a través de la librería *libmboard*, escrita usando la interfaz de paso de mensajes MPI. *Libmboard* realiza un sistema de coordinación y filtrado de los mensajes de los agentes antes de ser enviados a los agentes locales y a agentes de procesos externos. A través del paradigma Single Program Multiple Data (SPMD) este framework asegura que toda la información entre los agentes esté bien coordinada a través de puntos de sincronización.

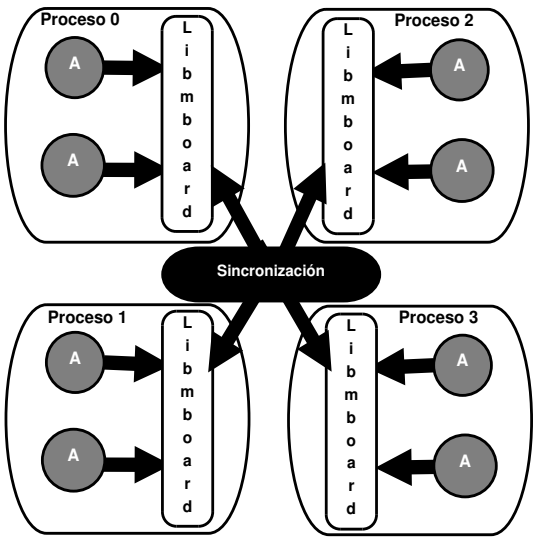

Fig. 3. Sincronización entre procesos a través de libmboard.

En la figura 3 se puede ver la comunicación entre agentes a través de la librería *libmboard*, la cual se encargará de enviar los mensajes a través de una comunicación sincronizada entre diferentes procesos MPI.

C. Distribución de Agentes

La distribución de los agentes en FLAME se puede realizar a través de una distribución estática geométrica o a través de una distribución estática round robin.

Actualmente FLAME no dispone de los mecanismos que permitan el movimiento de agentes entre procesos, por lo cual la carga de trabajo en cada proceso dependerá de la evolución del modelo a partir de los agentes iniciales. Por lo tanto, la propia

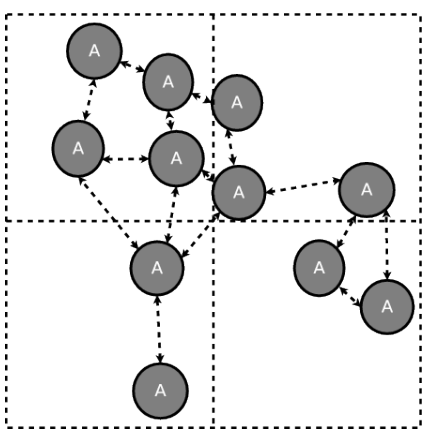

Fig. 4. Problema de la distribución.

evolución de los agentes puede desencadenar problemas de sobrecarga de cómputo y exceso de comunicaciones internodo debido a las interacciones entre agentes como se ve en la figura 4.

III. Migración de Agentes

Nuestra contribución consiste en brindar a FLAME la capacidad de generar automáticamente las funciones de migración de agentes entre procesos independiente del modelo a simular. Utilizando la misma metodologia de FLAME para la generación de código a través de plantillas se generan estas funciones. Estas plantillas son capaces de capturar la información necesaria del modelo para generar de forma automática las funciones necesarias para la migración agentes en conjunto con el código de simulación.

De manera general la migración de agentes está basada en mover los agentes que se desea migrar a *listas enlazadas de salida*, estas listas tienen las mismas características que las usadas para las transiciones y están identificadas por el id o rank del proceso de destino, posteriormente los agentes en las *listas de salidas* se empaquetan para ser enviados. De forma análoga en el proceso receptor se desempaquetan los datos recibidos y insertan con los demás agentes del proceso. Los algoritmos 1 y 2 reflejan los procedimientos involucrados durante la migración de agentes.

De la misma forma, como se explicó con anterioridad, FLAME almacena los agentes en listas enlazadas, por lo cual, para enviar los agentes a un destinatario mediante una sola comunicación se necesita almacenar los agentes en memoria contigua, lo cual se logra empaquetando los datos mediante MPI_Pack/MPI_Unpack. Ambas funciones requieren buffers de memoria antes de ser utilizadas, por lo cual las funciones que se generan calculan automáticamente los tamaños de buffer necesarios.

Algorithm 1 Envío de agentes

```
while agentes para enviar do
    remover del proceso actual
    insertar en la lista destino
end while
calcular tamaño de buffers de memoria
empaquetar listas
realizar envíos
```

Algorithm 2 Recepción de agentes

```
preguntar cuantos agentes se deben recibir
preparar buffers de memoria
realizar recepción
while agentes empaquetados do
    desempaquetar e insertar en el proceso actual
end while
```

La función de MPI_Pack empaqueta todos los agentes en un buffer de memoria contigua antes de ser enviados y de la misma manera MPI_Unpack después de recibir un buffer de datos contiguos desempaqueta los agentes.

Antes de realizar la migración se debe establecer un criterio para seleccionar los agentes a enviar y después a través de las funciones mencionadas en la sección III-B iniciar el proceso de migración. La migración también requerirá decidir cómo se realizarán los envíos y recepciones, esto dependerá en parte del criterio con el que se seleccionaron los agentes.

A continuación se explica cómo realizar la incorporación en FLAME para que sean generadas las funciones de migración en conjunto al código de simulación y se exponen las principales funciones de migración incorporadas a FLAME.

A. Inserción en FLAME

Para que la generación automática de las funciones de movimiento se lleve a cabo se deben realizar los siguientes pasos:

- Insertar las plantillas de movimientos de agentes al fichero xparser.c de FLAME para ser analizados sintácticamente de la misma forma que el resto de ficheros .tmpl.
- Insertar en la plantilla Makefile.tmpl la instrucción para generar los nuevos ficheros que contendrán las funciones de migración para ser compiladas junto con código de simulación
- Insertar en la plantilla main.tmpl la instrucción para habilitar libmboard en modo de acceso $MB\_MODE\_READWRITE$ en todos los procesos e insertar la función de inicialización (Opcionalmente, se puede realizar manualmente sobre el fichero generado).

B. Funciones de Migración

Las funciones se generarán de forma específica para cada tipo de agente en el modelo, lo que facilita la posibilidad de realizar migraciones después de cualquier transición. A continuación se presentan de forma global las principales funciones para la migración de agentes. NAME corresponde a nombre que describe al tipo de agente especifico.

- $Init\_movement$: Inicializa las estructuras necesarias para la migración.
- $prepare\_to\_move\_NAME$: Mueve agentes a la *lista de salida* correspondiente y lo remueve del actual proceso.
- $Pack\_NAME\_agent\_list$: Convierte todos los agentes de las *listas de salida* en datos del tipo MPI_PACKED, uno por cada destinatario.
- $send\_packed\_NAME/recv\_packed\_NAME$: Prototipos de rutinas para envio y recepción de agentes.
- $UnPack\_NAME\_agent\_list$: Desempaqueta los datos recibidos y los inserta como agente del proceso actual.

C. Utilización sobre el código de simulación

Una vez generado el código de simulación junto con las funciones de migración se deben insertar las

llamadas a las siguientes funciones.

- Insertar la llamada *init_movement*() antes de la sección de llamadas a la función $MB\_SetAccessMode$ para inicializar las estructuras necesarias para la migración.
- Poner todas las llamadas a $MB\_SetAccessMode$ en modo $MB\_MODE\_READWRITE$ para habilitar la comunicación entre agentes en todos los procesos.
- Seleccionar los agentes que se desean migrar junto con el destinatario a través de la función $prepare\_to\_move\_NAME$.
- Llamar a la función $Pack\_NAME\_agent\_list$ para empaquetar todos agentes.
- Escribir la rutina de comunicación para intercambiar los agentes usando $send\_packed\_NAME$ y $recv\_packed\_NAME$.
- Llamar a la función $UnPack\_NAME\_agent\_list$ para desempaquetar los agentes recibidos e insertarlos para continuar con la simulación.

Con estas funciones es posible incorporar a FLAME un mecanismo de balanceo de carga basado en la migración de agentes entre procesos.

IV. Caso de Uso

A continuación se presentan dos ejemplos con el objetivo de demostrar que mediante la utilización de las funciones de migración sobre el código generado por FLAME, se provee de la capacidad de mover agentes. El primer ejemplo correspondiente a un modelo de enfermedades SIR en un espacio toroidal 2D, a modo de ejemplo se decide migrar a los agentes infectados a un proceso en particular. El segundo caso corresponde a un modelo de proteinas compuesto por dos tipos principales de agentes, los cuales decidimos separar según su tipo en procesos diferentes a través las funciones de migración.

A. Modelo SIR

Este modelo describe evolución en el tiempo de la propagación de una enfermedad a través de una población de individuos de los cuales un pequeño grupo están infectados. El nombre de este modelo proviene de las tres clases de individuos con los que se trabaja: *Susceptibles* que pueden enfermarse, $Infectados$ que transmiten la enfermedad a los del tipo susceptible y *Recuperados* que se enfermaron y se inmunizaron. El modelo SIR utilizado considera nacimientos de individuos, por los cual la población susceptible a la enfermedad varia, también los individuos pueden morir por la enfermedad o de manera natural, en esta última forma incluye a todos los individuos. El nacimiento y la muerte de individuos representa la creación y eliminación de agentes en forma dinámica por lo que la carga de trabajo variará conforme avanza la simulación.

En este modelo cada individuo es un agentes de tipo `Person`, esto implica que muchas de las funciones generadas llevaran la etiqueta `Person`.

Siguiendo los pasos descritos en el punto III-C primero se debe insertar en el fichero `main.c` la llamada a la función *init_movement*() antes de la sección de configuración de acceso de libmboard (ver figura 5). Luego en cada sección de configuración de los mensajes de *libmboard*, se deben poner en modo lectura y escritura ($MB\_MODE\_READWRITE$) a través de una instrucción como la de la figura 6. En este modelo se realiza una comunicación de los infectados y su ubicación, por lo cual sólo se necesita configurar un tipo de mensaje ($FLAME\_infected\_message$).

```
init_movement();
/* For each message check if their exists agents
that input/output the message */
```

Fig. 5. Sección de inserción de la función de inicialización.

```
FLAME_infected_message_board_write = 1;
FLAME_infected_message_board_read = 1;
/* Call message board library with details */
```

Fig. 6. Habilitación del modo READWRITE de libmboard.

Estas dos primeras intervenciones en el fichero `main.c` pueden ser evitadas a través del uso de la plantilla modificada `main.tmpl`, ya que de esta manera serian generadas automáticamente en el fichero `main.c`. Posterior se debe implementar el criterio de selección para migrar los agentes, en este caso el criterio consiste en mover los agentes infectados al proceso 0 y los que no infectados al proceso 1. Como se muestra en la figura 7 primero se necesita un puntero para recorrer la lista de agentes almacenada en `Persons->agents`, luego se recorre la lista de agentes y todos aquellos en los cuales el atributo `is_sick` indiqué que están infectados se enviarán a la *lista de salida* de 0 a través de la función $Prepare\_to\_move\_Person$ la cual se encargará de remover el agente del actual proceso. De forma análoga, los agentes no infectados se enviarán a la *lista de salida* de 1. Finalmente se debe llamar a las funciones de empaquetado y desempaquetado, y en medio de ambas funciones se debe implementar algún método para intercambiar los agentes ya empaquetados. El código de la figura 7 ha sido insertado antes de la llamada a la función para escribir los estados de los agentes en disco (función *saveiterationdata*).

Para facilitar la visualización hemos decidido realizar una simulación con una población inicial de 150 agentes con un particionamiento geométrico en dos procesos MPI, y también el criterio de migración definido anteriormente lo aplicaremos a partir de la iteración 11.

En la figura 8 se muestra la evolución de la simulación en la iteración 10 a partir de una partición geométrica en la cual se aprecia a los agentes del proceso 0 con el símbolo ■ y los del proceso 1 con el símbolo +. Durante la iteración 11 se realizó la migración de todo los agentes infectados al proceso

```
xmachine_memory_Person_holder *current;
current = Persons->agents;
while(current){
 if(current->agent->is_sick == 1){
   if(node_number!=0)
     prepare_to_move_Person(0,&current,&Persons);
   else{
     current = current->next;
   }
 }else if(current->agent->is_sick == 0){
    if(node_number!=1)
      prepare_to_move_Person(1,&current,&Persons);
    else{
      current = current->next;
    }
 }
Pack_Person_agent_list();
exchange_Person_packs();
UnPack_Person_agent_list(Persons);
}
```

Fig. 7. Criterio de selección de agentes y posterior llamada a las funciones de migración.

0 y los agentes no infectados al proceso 1. Como resultado del proceso de migración se obtiene la distribución mostrada en la figura 9.

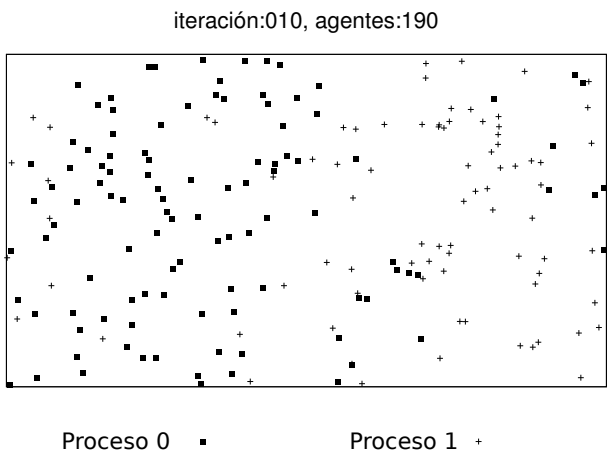

Fig. 8. Distribución sin separación infectados.

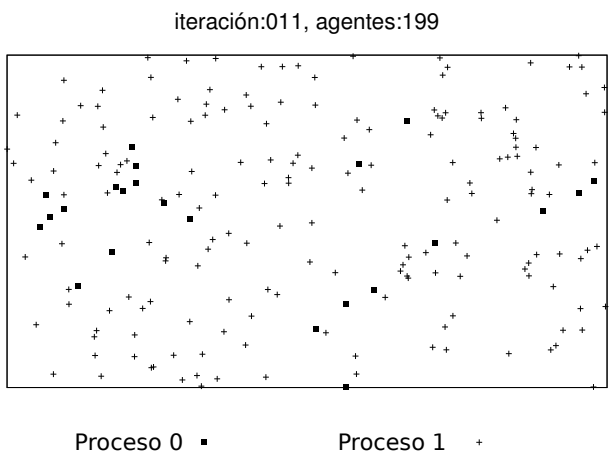

Fig. 9. Distribución después de la migración de infectados.

B. Modelo proteinas NF-kB

Los agentes en este modelo son moleculas claves y receptores asociasdos con una parte especifica de la celula, cada molécula se modela como un agente que puede moverse alrededor de la célula e interactúa con otras moléculas[7]. Este modelos posee dos tipos de agentes principales Proteinas y Receptores, y para este ejemplo en particular se migrarán los agentes tipo Proteina al proceso 1 y los agentes tipo Receptor al proceso 0.

Al igual que en el ejemplo anterior el objetivo de este experimento es demostrar que se pueden mover agentes. En la figura 10 se aprecia una distribución geométrica durante la iteracion 48, pero durante la iteracion 49 realizamos la migración de agentes segun el criterio del parrafo anterior.

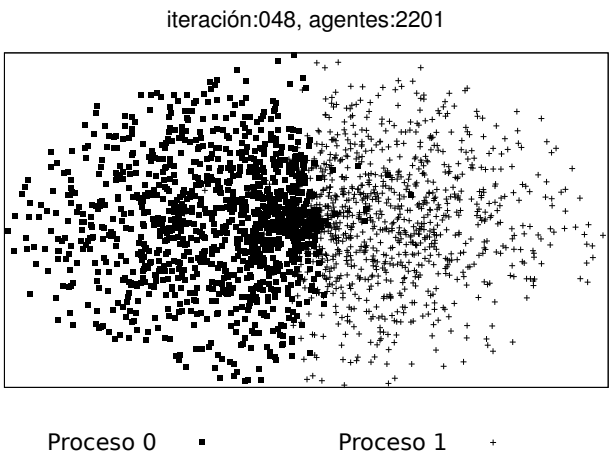

Fig. 10. Distribución sin migración.

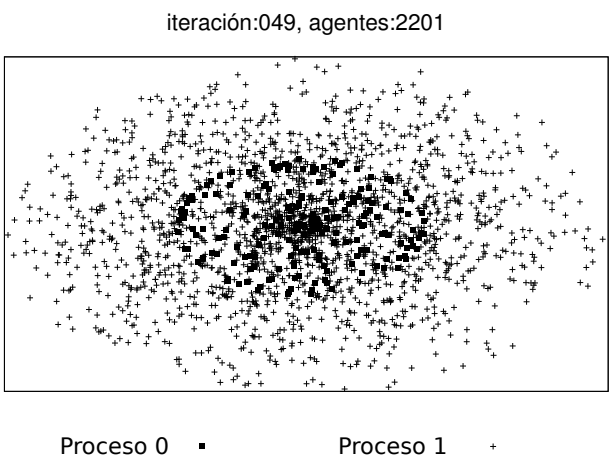

Fig. 11. Distribución después de la migración por tipos de agentes.

V. Conclusiones

Las aplicaciones ABMS presentan problemas de desbalanceo de cómputo y comunicación durante el proceso de simulación debido a los altos niveles de interacción entre agentes y a la diferente reglas de comportamiento. Por lo cual, para resolver este tipo de problemas las plataformas de simulación deberían estar dotadas de mecanismos de migración de agentes.

En este trabajo presentamos una modificación al framework FLAME que permiten realizar el movimiento de agentes entre procesos, con lo cual se pueden hacer modificaciones en la carga de trabajo entre diferentes procesos durante el proceso de simulación.

Esta mejora se ha realizado utilizando la misma metodología de FLAME para la generación de código a través de plantillas que generan de forma automática las funciones para migración independiente del modelo a simular, por lo que, también esta mejora permitirá en el futuro poder tomar decisiones automáticas de sintonización en cuanto a la carga de trabajo.

VI. Trabajos Futuros

Como trabajo futuro nos interesa analizar el rendimiento de FLAME a través de una herramienta de profiling y analizar políticas de balanceo de carga encontradas en problemas del tipo AMR[8](Adaptive Mesh Refinement) usando estas funciones de migración generadas. Consecuentemente con esto, queremos definir una metodología para de generalizar una estrategia de balance de carga y evaluar las propuestas para mejorar el rendimiento.

Agradecimientos

El presente trabajo ha sido financiado mediante MICINN España, bajo los proyectos TIN2007-64974 y TIN2011-28689-C02-01.

Referencias

[1] Rob Allan, "Survey of agent based modelling and simulation tools," *Engineering*, vol. 501, pp. 57–72, 2009.

[2] Russell K. Standish and Richard Leow, "Ecolab: Agent based modeling for c++ programmers," *CoRR*, vol. cs.MA/0401026, 2004.

[3] Russell K. Standish and Duraid Madina, "Classdesc and graphcode: support for scientific programming in c++," *CoRR*, vol. abs/cs/0610120, 2006.

[4] N.T. Collier and M.J. North, "Repast sc++: A platform for large-scale agent-based modeling," *Large-Scale Computing Techniques for Complex System Simulations*, 2011.

[5] Mariam Kiran, Paul Richmond, Mike Holcombe, Lee Shawn Chin, David Worth, and Chris Greenough, "Flame: simulating large populations of agents on parallel hardware architectures," in *AAMAS*, 2010, pp. 1633–1636.

[6] C. Deissenberg, S. Vanderhoog, and H. Dawid, "Eurace: A massively parallel agent-based model of the european economy," *Applied Mathematics and Computation*, vol. 204, no. 2, pp. 541–552, 2008.

[7] Mike Holcombe, Salem Adra, Mesude Bicak, Shawn Chin, Simon Coakley, Alison I. Graham, Jeffrey Green, Chris Greenough, Duncan Jackson, Mariam Kiran, Sheila MacNeil, Afsaneh Maleki-Dizaji, Phil McMinn, Mark Pogson, Robert Poole, Eva Qwarnstrom, Francis Ratnieks, Matthew D. Rolfe, Rod Smallwood, Tao Sun, and David Worth, "Modelling complex biological systems using an agent-based approach," *Integr. Biol.*, vol. 4, no. 4, pp. 53–64, 2012.

[8] Lori Freitag Diachin, Richard Hornung, Paul Plassmann, and Andy Wissink, *Parallel Adaptive Mesh Refinement*, chapter 8, pp. 143–162, Society for Industrial and Applied Mathematics, Philadephia, PA, 2006.