MARL-Ped+Hitmap: Aumentando la productividad de simulaciones basadas en agentes con una herramienta de arrays distribuidos

Eduardo Rodriguez-Gutiez [1], Francisco Martinez-Gil [2], Juan Manuel Orduña-Huertas [3], Arturo Gonzalez-Escribano [4]

Resumen— **Los sistemas Multi-agente están constituidos por piezas software llamadas agentes que son capaces de percibir el entorno y actuar en él de manera autónoma. MARL-Ped es un modelo Multi-agente de peatones donde cada agente (peatón) aprende el comportamiento adecuado para la simulación de diferentes situaciones (aglomeraciones, cruces, evacuaciones de recintos cerrados,...). MARL-Ped utiliza el estándar de paso de mensajes MPI para su explotación en sistemas distribuidos de forma portable. Programar utilizando directamente sistemas de paso de mensajes requiere un gran esfuerzo si se desean introducir políticas de reparto de carga flexibles y que se adapten a la plataforma de destino. Hitmap es una biblioteca de funciones para facilitar la programación de aplicaciones paralelas, basada en arrays distribuidos. Introduce abstracciones para la partición y mapeo transparente de arrays, así como para construir patrones de comunicación flexibles que se adaptan a la partición de forma automática.**

En este trabajo presentamos la metodología y técnicas de Hitmap aplicadas a la simulación de agentes, utilizando MARL-Ped como caso de estudio. Mostramos conceptual y experimentalmente las ventajas de aplicar Hitmap para aumentar la productividad de este tipo de aplicaciones, tanto en adaptabilidad como en rendimiento, permitiendo agrupar agentes en procesos y reduciendo los costes de comunicación y sobrecargas de forma transparente.

Palabras clave— **Agentes, simulación de multitudes, paso de mensajes, herramientas de programación, arrays distribuidos**

I. Introducción

Los sistemas Multi-agente [1], [2] están basados en la construcción de procesos computacionales independientes denominados agentes que tienen la capacidad de percibir el entorno y, de manera autónoma, tomar decisiones acerca de las actividades a llevar a cabo para conseguir sus objetivos. Este tipo de software es especialmente interesante para estudiar sistemas complejos, como la dinámica de los peatones, donde la interacción autónoma de los individuos genera comportamientos globales del sistema. MARL-Ped [3] es un modelo de peatones Multi-agente distribuido donde cada agente (peatón) aprende un comportamiento propio, que permite simular a grupos de peatones (desde unos pocos hasta multitudes) en diferentes escenarios como congestiones, cruces, avance en colas etc. La cantidad de agentes necesaria para simular muchedumbres, así como la gran carga computacional que implica tanto el proceso de aprendizaje de los agentes como la simulación de su comportamiento y del entorno, hacen de MARL-Ped un buen ejemplo de aplicación paralela apropiada para entornos de alto rendimiento. La versión actual de MARL-Ped utiliza el estándar de paso de mensajes MPI para su explotación en sistemas distribuidos de forma portable. Programar utilizando directamente sistemas de paso de mensajes requiere un gran esfuerzo si se desean introducir políticas de reparto de carga flexibles y que se adapten a la plataforma de destino.

Por otra parte, Hitmap [4] es una biblioteca de funciones diseñada para facilitar la programación de aplicaciones paralelas, basada en arrays distribuidos. Introduce abstracciones para la partición y mapeo automático de arrays, así como para construir patrones de comunicación flexibles que se adaptan a la partición de forma automática.

En este trabajo presentamos la metodología y técnicas de Hitmap aplicadas a la simulación de agentes, utilizando MARL-Ped como caso de estudio. Mostramos las ventajas de aplicar Hitmap para aumentar la productividad de este tipo de aplicaciones, tanto en esfuerzo de desarrollo, como en adaptabilidad y rendimiento. Los resultados muestran que la versión de MARL-Ped utilizando Hitmap es más simple en términos de código, y a la vez más flexible. Hitmap permite de forma transparente agrupar agentes en procesos para reducir los costes de comunicación y evitar sobrecargas, permitiendo una mayor escalabilidad y una explotación más eficiente de los recursos de cómputo.

El resto del artículo se estructura de la siguiente forma. La sección II presenta trabajo relacionado. La sección III describe los fundamentos de MARL-Ped y Hitmap. La sección IV describe la aplicación de Hitmap al programa original. La sección V presenta un estudio experimental en términos de esfuerzo de desarrollo y escalabilidad de la solución conseguida. Finalmente la sección VI presenta las conclusiones y trabajo futuro.

[1]Dpto. de Informática, Univ. Valladolid, e-mail: eduardo@infor.uva.es
[2]Dpto. de Informática, Univ. Valencia, e-mail: francisco.martinez-gil@uv.es
[3]Dpto. de Informática, Univ. Valencia, e-mail: juan.orduna@uv.es
[4]Dpto. de Informática, Univ. Valladolid, e-mail: arturo@infor.uva.es

II. Trabajo relacionado

Los estudios de dinámicas de peatones se han desarrollado a lo largo de los últimos 75 años. Aunque la aparición de los primeros modelos de peatones basados en modelos físicos de fluidos y las ecuaciones cinéticas de gases son de los años 60 del siglo XX, el verdadero desarrollo de modelos aconteció con el auge del uso de computadores de bajo coste a partir de la década de los 80 del pasado siglo. Entre los modelos de peatones que más éxito han tenido entre la comunidad científica debido a su versatilidad y simplicidad se encuentra el modelo de fuerzas sociales [5], los modelos basados en autómatas celulares [6], los modelos continuos basados en ecuaciones de cinética de gases [7] y los basados en agentes [8]. Dentro de este último grupo, en los últimos años se han realizado trabajos para incorporar técnicas de aprendizaje máquina (machine learning) a este campo [9]. Los sistemas de aprendizaje aplicado al modelado de peatones tiene características muy interesantes que los señalan como una alternativa a los sistemas más clásicos indicados anteriormente. Entre ellas la más destacada consiste en que es el propio agente o peatón el que aprende su comportamiento, liberando al programador de esta tarea. Siendo la dinámica de grupos de peatones un problema complejo, esta tarea se ha convertido en la dificultad principal de todo modelo de peatones.

Por otra parte, a nivel computacional la simulación microscópica de peatones en escenarios de grandes densidades (multitudes) constituyen un desafío en el que son necesarias nuevas estrategias de procesamiento paralelo y organización y paso de información. En este sentido, en el trabajo de Lozano et al. [10] se propone una arquitectura escalable basada en un sistema cliente-servidor jerárquico que soporta del orden de miles de agentes. En los trabajos posteriores [11], [12], [13] se propone una arquitectura paralela para simulación de muchedumbres en la que servidores interconectados comparten la carga computacional proveniente de dos cuellos de botella: el servidor de acciones y la base de datos que representa el terreno. En el trabajo de Yilmaz et al. [14] se propone una arquitectura basada en CUDA con lógica difusa que se usó para simular un maratón con más de un millón de corredores.

Hitmap ofrece una capa de abstracción intermedia entre el manejo manual en paso de mensajes de estructuras distribuidas y los lenguajes PGAS (Partitioned Global Adress Space), como Chapel [15], o UPC [16]. Estos modelos no proporcionan suficientes herramientas para permitir el paralelismo de procesos jerárquicos en entornos híbridos como Hitmap. Hitmap permite además construir patrones de comunicación reutilizables en tiempo de ejecución que se adaptan a la partición de datos, generando un número mínimo de comunicaciones agregadas. Esto permite conseguir, por ejemplo, una eficiencia comparable a UPC reduciendo incluso la complejidad de programación [4]. Hitmap se utiliza como capa de ejecución en el sistema de programación paralela

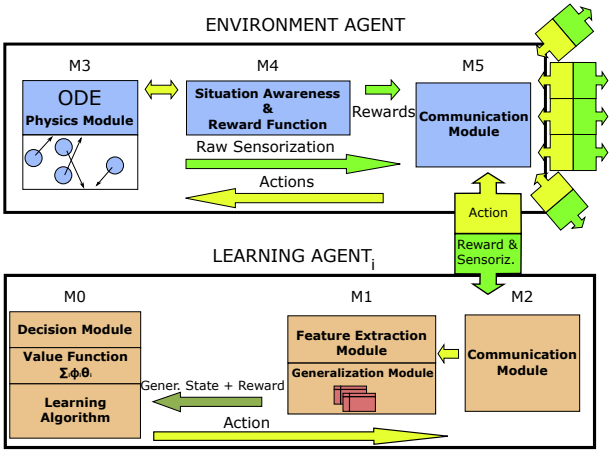

Fig. 1. Esquema de los tipos de agentes y su relación en MARL-Ped.

Trasgo [17], que ofrece una aproximación similar a los lenguajes PGAS. Hitmap extiende las funcionalidades de creación de jerarquías y de reparto de datos de otras bibliotecas o modelos de arrays distribuidos, por ejemplo HTAs [18] o Parray [19]. Entre otras cosas permite la utilización de políticas de partición transparentes e intercambiables, regulares o irregulares, definidas como módulos con un interfaz común. Esto elimina la necesidad de tomar decisiones sobre la granularidad y sincronización en diferentes niveles jerárquicos. Hitmap ha sido también extendido para soportar estructuras de datos como matrices dispersas o grafos, usando la misma metodología e interfaz [20].

III. MARL-Ped y Hitmap

A. MARL-Ped

MARL-Ped es un sistema multiagente para modelado y simulación de peatones que utiliza aprendizaje por refuerzo (reinforcement learning, RL) para aprender el comportamiento individual de cada peatón (agente) en un determinado escenario (congestión, evacuación de un recinto cerrado, circulación por un pasillo con flujos de movimiento contrarios,...). El objetivo del algoritmo de aprendizaje RL consiste en calcular una función de control (llamada función de valor en el campo de RL) que indicará al peatón qué acción tomar en cada momento en función del estado local percibido. Existen en MARL-Ped dos clases de agentes: los agentes peatón, que ejecutan los algoritmos de aprendizaje por refuerzo y almacenan la función de control aprendida, y el agente entorno que ejecuta el sistema físico que simula a los peatones en el escenario y que sensoriza el estado en el que se encuentra cada peatón dentro de este escenario. El escenario es un mundo virtual 3D en el que el motor físico Open Dynamic Engine (ODE) se encarga de simular las colisiones y las fuerzas que mueven a los peatones. En la Figura 1 aparece una descripción gráfica del sistema.

Es posible observar en la Figura 1 que tanto los agentes peatones (en la figura “Learning Agents”) como el agente entorno (en la figura “Environment

agent") están compuestos de diferentes módulos funcionales. Existen dos modos de funcionamiento en MARL-Ped: modo de aprendizaje y modo de simulación. Ambos modos establecen el mismo tipo de comunicación entre los agentes y el entorno. La única diferencia es que en el modo de aprendizaje los algoritmos de RL están activos y la función de control se va calculando incrementalmente.

El modo de funcionamiento es síncrono y consiste en un ciclo clásico compuesto de observación-acción-recompensa. Concretamente el ciclo se compone de los siguientes pasos:

1. El agente entorno consulta a ODE la situación dinámica de cada peatón que consiste en posición, velocidad, distancia a los n peatones más cercanos y distancia a los n objetos más cercanos. Si estamos en el modo aprendizaje, el agente entorno prepara una recompensa para cada uno de los agentes peatones en función de varios hechos:que el peatón haya llegado a su objetivo, que se haya chocado contra otro peatón u objeto, etc.
2. El agente entorno transmite la información del estado y la recompensa de cada agente al sistema para que sea recogida por los agentes.
3. Cada agente recibe la información del entorno y prepara con ella las características específicas que definen el estado para el algoritmo de aprendizaje y la señal de recompensa. Ambas informaciones son usadas por el algoritmo de RL para modificar la función de control que también está gestionada por el agente peatón ("Learning agent" en la Figura 1). En el modo de simulación, la información del estado sirve para consultar directamente ala función de control mientras que los algoritmos de RL están desactivados.
4. El agente consulta a la función de control la nueva acción que, en el estado actual, se debe de ejecutar. Las acciones consisten en una modificación de la dirección y rapidez con la que se mueve el peatón.
5. Los agentes transmiten las acciones al agente entorno que es el encargado de traducirlas a acciones físicas que son ejecutadas por ODE en el entorno virtual.

La comunicación de datos en MARL-Ped se establece entre el agente entorno y los agentes peatones, no habiendo comunicación entre los agentes peatones. Existe un doble flujo de datos, que se puede observar en la Figura 1. Un flujo comunica los datos de sensorización del estado de cada agente y la correspondiente recompensa desde el entorno hasta cada uno de los agentes (flechas de color verde). Otro flujo comunica la siguiente acción a realizar por cada uno de los agentes al entorno (flechas de color amarillo). Este ciclo descrito se ejecuta un número de veces que es a su vez un parámetro de configuración del sistema. En el modo aprendizaje con decenas de agentes, este ciclo puede repetirse desde cientos de miles de veces hasta millones de veces.

B. Hitmap

Hitmap [4] es una biblioteca de funciones para la gestión y distribución jerárquica de estructuras de datos en tiempo de ejecución. Trabaja sobre arrays densos y ha sido también extendida para soportar estructuras de datos como matrices dispersas o grafos, usando la misma metodología e interfaz [20]. Se basa en un modelo SPMD (Single Program Multiple Data) y en el paradigma de paso de mensajes. Hitmap define varias abstracciones para escribir programas paralelos que manejan estructuras de datos distribuidas. La biblioteca se puede dividir en tres partes:

B.1 Métodos de tiling

Definición y manipulación de arrays y tiles. Estos métodos se pueden usar de manera independiente al resto de funcionalidades de Hitmap, para mejorar la localidad en el código secuencial, así como para generar distribuciones de datos manualmente para la ejecución en paralelo. Se definen objetos para representar los dominios de índices de las estructuras de datos de forma compacta. Se define una clase de objetos denominada *Tile* que representa la asociación entre elementos del espacio de índices y los datos, y permiten el acceso o modificación de los mismos. Los tiles pueden crearse crearse de forma que en un proceso sólo se dispone localmente de una parte del espacio de índices obteniéndose la estructura distribuida.

B.2 Métodos de mapping

Incluye módulos intercambiables que implementan políticas para particionar dominios automáticamente, dependiendo de una topología virtual seleccionada en tiempo de ejecución. Permiten crear objetos denominados *Layouts* que pueden ser consultados respecto a la parte de un dominio de índices que es asignada tanto localmente, como o a cualquier otro proceso remoto. Además, se establecen relaciones de vecindad en términos de las reglas de la topología virtual escogida.

B.3 Métodos de comunicaciones

Estas funciones son una abstracción del modelo de paso de mensajes para comunicar tiles o partes de tiles entre procesadores virtuales. Permite crear objetos que almacenan la información necesaria para empaquetar e intercambiar los datos seleccionados entre procesadores. Hitmap provee de múltiples interfaces de creación que implementan diversos tipos de comunicaciones punto-a-punto, colectivas, o patrones más complejos compuestos de múltiples comunicaciones de diversa naturaleza o sobre diversos tiles. Las interfaces de creación incluyen un objeto de tipo *Layout*, que es consultado internamente para decidir qué tiene que comunicarse y a quién, generándose automáticamente la información interna necesaria. Por tanto, estos objetos se adaptan transparentemente en el momento de su creación a la plataforma de ejecución y la distribución de datos

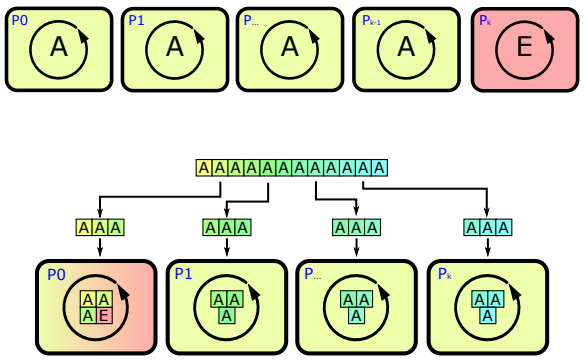

Fig. 2. Estructura general de la simulación y el reparto de tareas entre procesadores en MARL-Ped original (arriba) y tras aplicar Hitmap (abajo).

utilizada. Los objetos de comunicación son reutilizables. El método que realmente ejecuta la comunicación se puede invocar en cualquier momento después de su creación, tantas veces como sea necesario a lo largo de la aplicación. Internamente se basan en el estándar MPI, utilizando técnicas eficientes como la creación de tipos de datos derivados, comunicaciones asíncronas, etc.

IV. Aplicación de la metodología y técnicas de Hitmap

En esta sección se describe cómo aplicar la metodología y técnicas propias de Hitmap a aplicaciones de simulación basadas en agentes, utilizando MARL-Ped como caso de estudio.

A. Cambios estructurales

La estructura de la aplicación MARL-Ped ha sido rediseñada. La versión de Hitmap aplica el concepto de arrays distribuidos para agrupar agentes en procesos, en lugar de utilizar un proceso MPI para cada agente, más otro para la simulación del entorno. En la parte superior de la figura 2 se aprecia la distribución conceptual de la computación en MARL-Ped original. Cada proceso ejecuta el código de un agente. El último proceso ejecuta el cómputo de la simulación del entorno. Los objetos de clase *RLAgent* y *RLEnvironment* tienen un método que internamente ejecuta repetitivamente el bucle de simulación.

Una primera decisión de diseño para la nueva versión es repartir los agentes entre los procesos disponibles sin reservar un proceso para el entorno. El código del entorno lo ejecutará uno de los procesos que también tiene agentes asignados ya que la computación principal de los agentes y del entorno se alternan sin solaparse en el tiempo.

Hitmap provee de las herramientas necesarias para distribuir equitativamente los agentes entre los procesos disponibles (ver la parte de abajo en la figura 2). Cada proceso tiene que poder ejecutar en cada iteración de simulación el código de varios agentes, y en el caso del proceso designado para ello, además ejecutar el código del entorno. Por tanto, el bucle de simulación no puede estar dentro de los objetos de agente o entorno. Es necesario rediseñar la aplicación para que el bucle de simulación lo ejecute el programa principal, que debe iterar a su vez dependiendo del número de agentes asignados al proceso. Para ello se elimina el bucle de simulación dentro del correspondiente método de la clase de agentes y entorno. Se transforman los métodos a los que se invocaba dentro de ese bucle en métodos públicos. Y se traslada la lógica de control de dichas invocaciones al nuevo bucle de iteraciones de simulación en el programa principal. Se realiza lo mismo con el entorno, rodeando la lógica de control del mismo, ahora en el programa principal, con un condicional para que sólo sea ejecutado por un proceso. Hitmap distingue para cada grupo de procesos a uno de ellos como el líder del grupo. Este proceso puede identificarse a si mismo a través de una llamada a una función, y por tanto es el seleccionado para ejecutar la lógica del entorno.

B. Arrays distribuidos y patrones de comunicación

Las comunicaciones basadas en MPI de MARL-Ped original se han substituido por el manejo de arrays distribuidos con Hitmap. Para ello, las estructuras de datos implicadas en las comunicaciones se substituyen por estructuras de tipo *HitTile*. La estructura HitTile debe especializarse al comienzo del programa según los diferentes tipos de datos de cada array que se vaya a declarar y manejar con Hitmap.

En fase de inicialización del programa se crean los arrays distribuidos y objetos de tipo *HitCom* con las especificaciones de las comunicaciones que serán invocadas en las iteraciones del bucle de simulación. Para las señales de control sólo es necesaria una variable de tipo entero en cada proceso, independientemente del número de agentes que se le asignen. Para cada flujo de datos entre el agente que gestiona el entorno y los agentes peaton, se declaran dos arrays distribuidos con un dominio de índices igual al número de agentes peatón. En uno se utiliza una política de distribución que reparte y asigna los elementos de forma equitativa entre los procesos. En el otro se utiliza una política que distribuye todos los elementos al proceso que ejecuta el entorno. Hitmap permite con una única llamada a una función construir un objeto *HitCom* que implementa un patrón que redistribuye los datos desde un array distribuido con una política cualquiera, a los correspondientes elementos locales o remotos de otro array con el mismo dominio, pero distribuido con otra política diferente. Esta técnica permite construir objetos de comunicación que moverán de forma transparente los datos entre las dos copias de cada array, la realmente distribuida y la que tiene todo el dominio en el proceso del entorno. El patrón de comunicación se adapta a los resultados de las políticas de partición independientemente del número de agentes y procesos. Este mecanismo soluciona de una forma única la construcción de los flujos de comunicación necesarios.

V. Estudio experimental

Esta sección describe el estudio experimental realizado para comprobar las ventajas de aplicar Hitmap en programas de simulación basados en agentes como MARL-Ped. El estudio se focaliza en mostrar la mayor capacidad de escalado en cuanto a número de agentes, y la predictibilidad de los parámetros de ejecución para obtener un mayor rendimiento.

A. Metodología de experimentación

En este estudio experimental se obtienen medidas de tiempos de ejecución de las dos versiones de código; MARL-Ped original y la versión utilizando Hitmap. Se presentan los mismos en términos de escalabilidad.

En este artículo nos focalizamos en la parte del proceso de aprendizaje, que es computacionalmente la más costosa, y que no implica operaciones de entrada/salida durante la fase de computación/comunicación. Los códigos han sido instrumentados para medir el tiempo de ejecución de cada proceso distribuido desde el momento en que comienza la inicialización de estructuras relacionadas con el paralelismo (MPI o Hitmap), hasta el momento en que termina la ejecución del aprendizaje, justo antes de comenzar a escribir en ficheros los resultados. De las mediciones obtenidas en cada proceso, se utiliza como resultado de la medida el tiempo mayor, es decir, el del proceso que ha tardado más en terminar. Cada experimento se repite varias veces para poder comprobar también la variabilidad en los resultados.

Dado que los tiempos de ejecución de un entrenamiento completo son extremadamente largos, para poder explorar un espacio de búsqueda amplio en cuanto a parámetros de ejecución, se ha limitado el programa a la ejecución de sólo 100 iteraciones de entrenamiento en todos los casos. Como se verá en los resultados, este número de iteraciones produce una carga y un número de fases de comunicación y sincronización suficientemente representativos. En todos los casos se ejecuta un escenario de ejemplo utilizado y validado en trabajos anteriores [3]. El escenario reproduce un dilema de navegación clásico en dinámica de peatones denominado "shortest path *vs* quickest path". En este escenario un grupo de peatones debe pasar de una habitación a un lugar objetivo situado fuera de ésta. Existen dos salidas y una está mas cerca del objetivo que la otra. Los agentes deben aprender que si todos intentan salir por la puerta más cercana, se formará una aglomeración que ralentizará la evacuación. Una solución óptima es la división del grupo en peatones que eligen la salida con el camino más corto y agentes que eligen la otra salida, lo que proporcionará una evacuación más rápida.

En la configuración seleccionada se han situado 28 agentes en una habitación rectangular de longitud de 18 m. con dos salidas posibles de 1 m.de anchura que dificulta el paso simultáneo de más de un peatón. El objetivo de los agentes es alcanzar un punto de encuentro al otro lado de las salidas.

Los códigos se han ejecutado en diferentes plataformas multicore donde las comunicaciones son menos costosas y destacan más los potenciales overheads asociados, entre otras cosas, a los cambios en la estructura de ejecución, al manejo de las estructuras propias de Hitmap, o a los cálculos y decisiones sobre las comunicaciones.

En este trabajo mostramos resultados obtenidos en una máquina denominada Chimera. Tiene dos CPUs Intel E5-2620 v2, a 2i̇0 GHz, con un total de 12 cores reales, con la opción de *hyperthreading* activada. Dispone de 8 Gb de memoria DDR4. El sistema operativo es CentOS 7.0 1406 x64. El compilador utilizado es GCC 4.8.2 con el flag de optimización -O3. La implementación de MPI utilizada es Mpich 3.1.3.

B. Escalabilidad en número de agentes

Uno de los objetivos de este trabajo es conseguir una mayor escalabilidad en términos de número de agentes, gracias a la estrategia de asociar varios agentes a un mismo proceso con Hitmap. El primer estudio realizado comprueba experimentalmente la diferencia de escalabilidad entre la versión original de MARL-Ped y la versión que utiliza Hitmap. Los programas se han ejecutado en fase de entrenamiento fijando el número de procesos MPI al número de cores disponibles en la máquina, aumentando progresivamente el número de agentes por encima de ese número.

Los resultados se muestran en la figura 3. En el caso de MARL-Ped original, el número de procesos MPI necesarios crece con el número de agentes (siendo el número de agentes más uno). La estructura de estas aplicaciones de simulación, que utilizan comunicaciones colectivas con puntos claros de sincronización global alrededor de la ejecución del motor de simulación, no presentan la propiedad de *parallel slackness*, que aparece en ciertas aplicaciones cuando varios procesos asignados al mismo elemento de proceso solapan alternativamente fases de computación y comunicación. En los resultados se observa cómo el sobrecoste de realizar *oversubscribing* (lanzar más procesos que elementos de proceso disponibles) tiene un impacto negativo en el rendimiento y la escalabilidad de la aplicación. En la máquina Chimera, con 12 cores y que tiene activado *hyperthreading*, el efecto es mucho más notable a partir de 24 agentes, donde MARL-Ped ya utiliza 25 procesos MPI. La forma en los que los procesos se rotan en los planificadores adquiere relevancia. Por ello se observan mayores diferencias entre los tiempos de ejecución mínimos y máximos. La versión que utiliza Hitmap, mantiene una buena escalabilidad, ya que limita el número de procesos y ejecuta secuencialmente dentro de cada uno el código de varios agentes de una forma más eficiente.

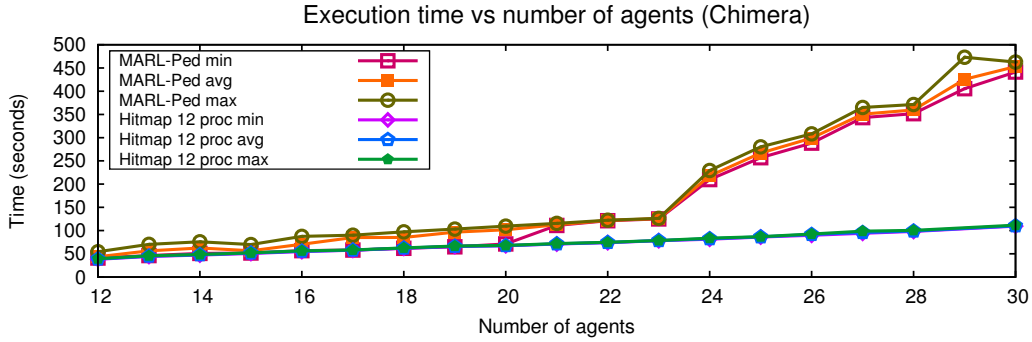

Fig. 3. Tiempos de ejecución en relación al número de agentes. La gráfica muestra el tiempo de ejecución de 100 iteraciones de entrenamiento para un número creciente de agentes. En el caso MARL-Ped original el número de procesos crece con el número de agentes. En el caso de MARL-Ped+Hitmap el número de procesos MPI es fijo, e igual al número de elementos de proceso (cores) reales disponibles.

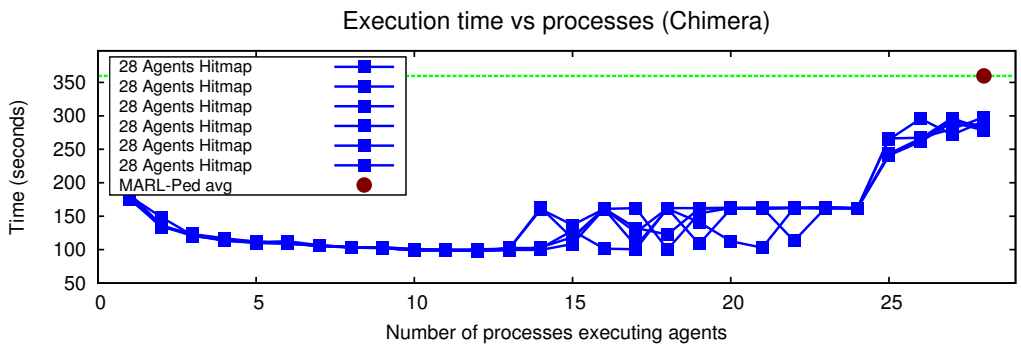

Fig. 4. Tiempos de ejecución en relación al número de procesos para un número de agentes fijo. La gráfica muestra el tiempo de ejecución de 100 iteraciones de entrenamiento. En el caso de MARL-Ped original el número de procesos MPI depende por diseño del número de agentes y es igual a 29. Para MARL-Ped+Hitmap se muestran varias líneas correspondientes a varios experimentos consecutivos, para mostrar la variabilidad de los resultados.

C. Impacto del número de procesos

En esta sección se estudia el impacto de modificar el número de procesos, y por tanto la distribución de agentes por proceso, en MARL-Ped+Hitmap. En la figura 4 se observan los resultados obtenidos para 28 agentes. En todos los casos los resultados en tiempos de ejecución de MARL-Ped+Hitmap son mejores que en MARL-Ped original debido al efecto de oversubscription comentado en la sección anterior.

Se observa que los resultados utilizando Hitmap mejoran ligeramente al ir repartiendo los agentes entre un número pequeño pero creciente de procesos, ya que aumenta el paralelismo. Al superar el número de procesos al número de cores reales (sin contar con hyperthreading), los resultados empeoran y se vuelven más inestables, ya que aparecen efectos negativos derivados del oversubscripting. Las políticas de planificación comienzan también a hacer los resultados más impredecibles. En algunos casos se obtiene tiempos tan buenos como antes de comenzar el oversubscripting, pero en muchos oros casos empeoran. Al superar el límite del número de threads disponibles contando con hyperthreading los resultados, como era de esperar empeoran notablemente.

Estos resultados indican que en MARL-Ped+Hitmap el número de procesos a escoger para el entrenamiento es predecible. Los tiempos de ejecución son menores y más estables cuando se utiliza un número de procesos igual al número de elementos de proceso reales, sin tener en cuenta la opción de hyperthreading.

VI. Conclusiones

En este artículo se presenta la aplicación de las técnicas y herramientas de la biblioteca de manejo de array distribuidos Hitmap a la simulación basada en agentes. Se utiliza la aplicación multi-agente para simulación de peatones MARL-Ped como caso de estudio. Se muestra cómo la utilización de arrays distribuidos y políticas de partición automáticas permite obtener una mayor productividad y escalabilidad, gracias a la capacidad de Hitmap de distribuir la carga entre procesos de forma transparente al programador. MARL-Ped+Hitmap permite hacer simulaciones con un número mucho mayor de agentes manteniendo tiempos de ejecución estables y predecibles.

El trabajo futuro incluye extender la aplicación de Hitmap a otros tipos de aplicaciones de simulación relacionadas; investigar la potencial eliminación de cuellos de botella en las fases de simulación; y utilizar la nueva versión de MARL-Ped+Hitmap para profundizar en el estudio de la calidad y resultados de las simulaciones de muchedumbres con un número mucho mayor de agentes.

Agradecimientos

Esta investigación ha sido parcialmente financiada por el Ministerio de Economía y Competitividad (Spain) y el programa ERDF de la Unión Europea: Proyecto HomProg-HetSys TIN2014-58876-P, Proyecto TIN2015-66972-C5-5-R y COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

Referencias

[1] M.J. Wooldridge and N.R. Jennings, "Intelligent agents: theory and practice," *The Knowledge Engineering Review*, vol. 10, pp. 115–152, 1995.

[2] M. Wooldridge, *Multi-Agent Systems 2nd Edition*, chapter Intelligent Agents, pp. 3–50, MIT Press, 2013.

[3] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández, "MARL-ped: A multi-agent reinforcement learning based framework to simulate pedestrian groups," *Simulation Modelling Practice and Theory*, vol. 47, pp. 259–275, 2014.

[4] A. Gonzalez-Escribano, Y. Torres, J. Fresno, and D.R. Llanos, "An extensible system for multilevel automatic data partition and mapping," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1145–1154, 2014.

[5] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Phys. Rev. E*, vol. 51, pp. 4282–4286, 1995.

[6] V. J. Blue and J. L. Adler, "Cellular automata microsimulation for modeling bi-directional pedestrian walkways," *Transportation Research Part B: Methodological*, vol. 35, no. 3, pp. 293–312, 2001.

[7] R. L. Hughes, "The flow of human crowds," *Annu. Rev. Fluid Mech.*, vol. 35, pp. 169–182, 2003.

[8] C. Reynolds, "Steering behaviors for autonomous characters," in *Game Developers Conference*, San Francisco, California., 1999, pp. 763–782, Miller Freeman Game Group.

[9] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández, "Multi-agent reinforcement learning for simulating pedestrian navigation," in *Adaptive and Learning Agents: International Workshop, ALA 2011, Held at AAMAS 2011, Taipei, Taiwan.* 2012, pp. 54–69, Springer Berlin Heidelberg.

[10] M. Lozano, P. Morillo, J. M. Orduña, V. Cavero, and G. Vigueras, "A new system architecture for crowd simulation," *J. Network and Computer Applications*, vol. 32, no. 2, pp. 474–482, 2009.

[11] G. Vigueras, M. Lozano, J. M. Orduña, and F. Grimaldo, "A comparative study of partitioning methods for crowd simulations," *Appl. Soft Comput.*, vol. 10, no. 1, pp. 225–235, Jan. 2010.

[12] Guillermo Vigueras, Juan M. Orduña, and Miguel Lozano, "A read-copy update based parallel server for distributed crowd simulations," *The Journal of Supercomputing*, vol. 64, no. 1, pp. 156–166, 2013.

[13] Guillermo Vigueras, Juan M. Orduña, Miguel Lozano, and Yvon Jégou, "A scalable multiagent system architecture for interactive applications," *Science of Computer Programming*, vol. 78, no. 6, pp. 715 – 724, 2013, Special section: The Programming Languages track at the 26th {ACM} Symposium on Applied Computing (SAC 2011); Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.

[14] E. Yilmaz, V. Isler, and Y. Y. Cetin, "The virtual marathon: Parallel computing supports crowd simulations," *IEEE Computer Graphics and Applications*, vol. 29, no. 4, pp. 26–33, 2009.

[15] B.L. Chamberlain, D. Callahan, and H.P. Zima, "Parallel programmability and the chapel language," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 3, pp. 291–312, aug 2007.

[16] D. A. Mallón, A. Gómez, J. C. Mouriño, G. L. Taboada, C. Teijeiro, J. Touriño, B. B. Fraguela, R. Doallo, and B. Wibecan, "Upc performance evaluation on a multicore system," in *Proceedings of the Third Conference on Partitioned Global Address Space Programing Models*, New York, NY, USA, 2009, PGAS '09, pp. 9:1–9:7, ACM.

[17] Arturo Gonzalez-Escribano and Diego R. Llanos, "Trasgo: a nested-parallel programming system," *The Journal of Supercomputing*, vol. 58, no. 2, pp. 226–234, 2011.

[18] Basilio B. Fraguela, Ganesh Bikshandi, Jia Guo, MaríA J. GarzaráN, David Padua, and Christoph Von Praun, "Optimization techniques for efficient hta programs," *Parallel Comput.*, vol. 38, no. 9, pp. 465–484, sep 2012.

[19] Yifeng Chen, Xiang Cui, and Hong Mei, "Parray: A unifying array representation for heterogeneous parallelism," *SIGPLAN Not.*, vol. 47, no. 8, pp. 171–180, feb 2012.

[20] J. Fresno, A. Gonzalez-Escribano, and D.R. Llanos, "Blending extensibility and performance in dense and sparse parallel data management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2509–2519, 2014.