SIMULACIÓN USANDO AUTÓMATAS CELULARES CON MULTIPROCESAMIENTO

A PREPRINT

Rios Gastón Gustavo
Facultad de Informática
Universidad Nacional de La Plata
okason1997@hotmail.com

March 20, 2019

ABSTRACT

Se realizó el análisis de autómatas celulares y su implementación en la simulacion de fenomenos naturales, en mi caso elegí la simulación de las células Dictyostelea. Se compararon las posibles implementaciones utilizando pasaje de mensajes (sincrónicos y asincrónicos) con las de memoria compartida. Finalmente se realizó el software y se analizaron los resultados obtenidos.

1 Resumen del trabajo

Un autómata celular es un sistema computacional discreto compuesto por un número finito de unidades simples homogéneas llamadas células o átomos. En cada unidad de tiempo, las células poseen un estado, el cual cambiará en todas en paralelo en cada paso mediante una función de actualización, la cual tendrá en cuenta el estado de las celdas vecinas. Este tipo de comportamiento favorece en gran medida la paralelización de sus células. Los autómatas celulares tienen múltiples usos posibles, entre ellos la simulación de sistemas dinámicos. En mi caso elegí simular las células llamadas Dictyostelea. Estas son un grupo de protistas conocidos comúnmente como hongos mucilaginosos. Estos organismos pueden tomar tres etapas en su vida: una etapa unicelular, una etapa de agregación y una etapa de cuerpo fructífero formador de esporas. Esto le permite pasar de ser un organismo unicelular a multicelular. Este comportamiento lo realiza mediante la interacción con sustancias químicas del entorno midiendo su concentración para decidir qué camino tomar. Este comportamiento será la función de actualización de las células del autómata celular, donde cada célula, midiendo la concentración de sustancias químicas a su alrededor decidirá dónde moverse.

Queda clara la posibilidad de paralelizar los autómatas celulares, ya que se poseen múltiples células realizando la misma operación en paralelo en una unidad de tiempo. La complejidad se encuentra en que las células deberán comparar todas contra la misma matriz, la cual será modificada en cada paso. También hay que tener en cuenta la colisión de células, al moverse en paralelo y siguiendo la concentración de sustancia es muy frecuente la colisión de células, lo que podría causar la pérdida de alguna célula de no ser tratado el caso. En mi caso decidí que en caso de colisión una de las células no se mueva o busque modificar su camino para moverse hacia el químico.

El programa consiste de células ubicadas en un mundo, el cual es una matriz. En este mundo tambien hay comida, la cual estara rodeada de sustancias quimicas que atraen a las celulas a esta. Las células se moverán por el mundo buscando toda la comida posible para alimentarse. A medida que se mueven, las células irán gastando energía y cuando esta se agote emitieron una señal química a las demás células indicando una concentración baja de comida, las células que reciban esta señal comenzarán la etapa de agregación, en la cual emitirán la misma señal química que recibieron y se moverán hacia la mayor concentración de este químico para formar la etapa de cuerpo fructífero formador de esporas.

Se utilizó memoria compartida, lo cual le otorga beneficios a la hora de acceder para lectura de los datos de la matriz, ya que permite el acceso a los usando el manejo de bloques de caché del hardware, a diferencia de pasaje de mensajes donde sería necesario copiar la matriz para cada proceso. Para la sincronización se utilizó otra matriz, la cual poseerá las siguientes posiciones de las células a medida que se procesan, de esta forma las células podrán saber cuando colisionan y modificar su comportamiento acordemente.

El paralelismo logró reducir en gran medida el tiempo necesario. En la sección de resultados es posible ver la reducción lograda.

2 Descripción del problema

La memoria compartida implica que todos los procesos comparten una misma memoria y tienen acceso a esta. En memoria compartida la comunicación es implicita, los procesos se comunican mediante la modificación de variables compartidas, las cuales podrán ser accedidas por todos los procesos. El acceso a estas variables se realiza con las operaciones convencionales de acceso a memoria.

Este modelo tiene la desventaja de que al tener múltiples procesos con accesos de lectura y escritura a los mismos datos en memoria es necesario implementar algún tipo de sincronización, ya que en caso contrario podría producirse un intercalado de operaciones erróneo. El intercalado es el orden que toman las operaciones atómicas de los procesos al ejecutarse. Un orden incorrecto podría generar inconsistencia en los datos. Esta sincronización produce overhead, por lo que habrá que tenerla en cuenta, una sincronización incorrecta podría generar deadlock, inanición, inconsistencia de datos o una reducción en el tiempo de ejecución.

Existen diversas arquitecturas que apoyan la memoria compartida entre procesos. En configuración SMP (symmetric multi-processor) los procesadores están conectados a la misma memoria a través de un bus y todos poseen el mismo tiempo de acceso a todas las áreas de memoria. El problema de SMP es que, a pesar de funcionar bien con sistemas con pocos procesadores, al aumentar la cantidad de procesadores se satura el bus, ya que este posee una ancho de banda fijo y solo un procesador puede acceder a este al mismo tiempo, generando latencia en los demás procesadores que deben esperar para poder utilizarlo. Una memoria caché en cada procesador permite aligerar el tiempo de espera del bus, pero no soluciona del todo el problema.

Una alternativa a SMP son las arquitecturas NUMA (non-uniform memory access) en la cuales cada procesador posee una memoria local mas rapida y un controlador que permitirá acceder a las memorias de los demás procesadores, pero a una velocidad inferior. NUMA permite reducir la demanda de ancho de banda del bus, ya que cada proceso podrá tener acceso a sus instrucciones, stack y datos generalmente de forma local, junto a datos compartidos que tambien podrian estar de forma local al proceso, lo cual permite reducir los tiempos de acceso.

Para mantener consistencia entre los procesadores del sistema, se utiliza un protocolo de coherencia de cache por hardware, el cual se encarga que los accesos a la caché de los diferentes procesos no se vuelvan inconsistentes cuando uno de los procesos modifique un dato compartido, esto lo hace invalidando las copias obsoletas del dato en los cache de los demás procesadores, que para poder acceder al dato deberán realizar una lectura en memoria, lo cual aumenta la latencia, principalmente en arquitecturas NUMA si el dato se encuentra en una memoria no local.

El pasaje de mensajes permite la sincronización de los procesadores enviando mensajes entre los procesos del programa de forma sincrónica (bloqueante) o asincrónica (no bloqueante). El pasaje de mensajes se realiza de manera explícita, se utilizan operaciones send y receive para realizar la comunicación, esto simplifica la sincronización entre procesos. La conexión entre procesos está integrada a nivel de entrada/salida, a diferencia de memoria compartida en donde esta conexión se encontraba a nivel de acceso a memoria. Cada nodo en pasaje de mensajes funciona como una computadora diferente, con su propia memoria y procesador.

El intercambio de datos se produce memoria a memoria copiando los datos enviados. Es posible la utilización de DMA (direct memory access), un hardware especializado que permite la enviada no bloqueante de mensajes. El proceso que envíe el mensaje lo cargara en el buffer del DMA local, quien se encargará de enviarlo al receptor. En el lado receptor, el DMA del receptor cargará en su buffer el mensaje recibido hasta que el receptor realice la lectura correspondiente.

Para realizar un cambio a una variable compartida en pasaje de mensajes se utiliza un servidor, al cual los clientes le envían mediante un llamado a procedimiento remoto envían los cambios a realizar. Este servidor tendrá en su memoria local e idealmente en su caché los datos compartidos, por lo que su tiempo de acceso no cambia demasiado al aumentar la cantidad de procesos, aumentará proporcionalmente a la cantidad de clientes que le envíen mensajes a medida que se llena su caché de recibida.

Al aumentar en gran medida la cantidad de procesadores es conveniente el uso de pasaje de mensajes, ya que el uso de un servidor con los datos compartidos evita la generación de cache misses, los cuales escalan en complejidad a medida que aumenta la cantidad de procesadores que comparten datos en sus caches.

A pesar de esto, la transferencia de datos entre procesos de memoria compartida es más veloz que la transferencia entre procesos con pasaje de mensajes, ya que en pasaje de mensajes se deben copiar los datos a ser enviados y se posee mayor overhead, en cambio en pasaje de mensajes con escribir los datos en memoria basta, ya que el otro proceso también tiene acceso a estos, pero hay que tener en cuenta la sincronización.

En el problema dado, se posee una matriz compartida entre los procesos, el mundo del autómata celular que es donde están ubicadas las células. Este mundo irá cambiando en cada paso de actualización, y cada célula Dictyostelea deberá estar al tanto de las células que la rodean. Será necesario tener en cuenta la colisión de las células, ya que el método que se tome afectará el resultado final de la simulación. Se pueden tomar 2 acercamientos cuando ocurra una colisión entre células:

1. Choque: las células que intenten moverse al mismo lugar chocaran, lo que genera que solo una de las dos efectivamente realice el movimiento, mientras que la otra se quedará en su lugar.
2. Desliz: cuando dos células intenten moverse al mismo lugar, una de las dos se moverá al lugar, mientras que la otra buscará un camino alternativo que la acerque al lugar.

Figure 1: Mundo final luego del procesamiento utilizando choque

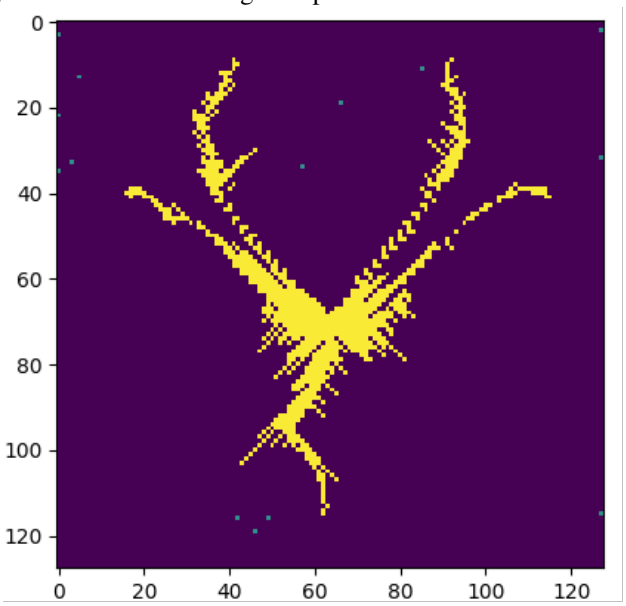
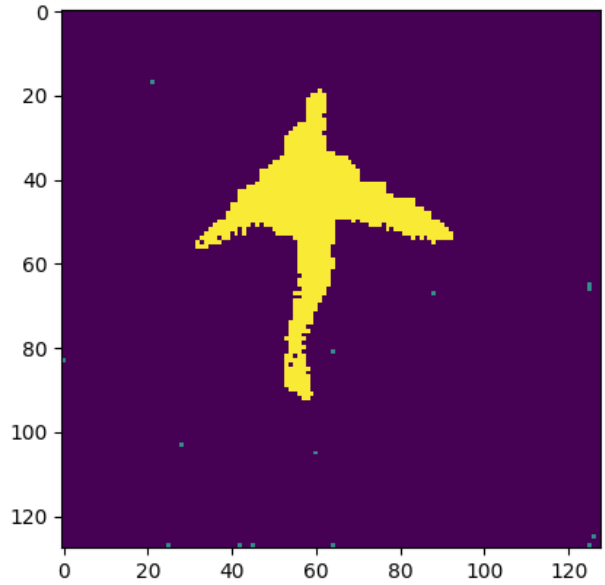


Figure 2: Mundo final luego del procesamiento utilizando desliz



Se puede observar en la imagen que al realizar la decisión de choque basada en movimiento, las células se traban entre sí, creando largas hileras de células que esperan que la siguiente se mueva hacia el centro del grupo de células agregadas.

En caso de utilizar desliz, se ve cómo las células encuentran una forma mejor de acercarse al centro de la agregación de células, creando una forma más natural. Es por esto que se decidió tomar este acercamiento.

Otro elección a tomar es la asincronía o sincronía del autómata celular. La asincronía consta de tomar de a una célula al azar del conjunto de células y ejecutar su función de actualización, manteniendo las demás células su anterior estado. La actualizacion sincronica consta de actualizar todas las celulas del mundo al mismo tiempo, utilizando la información obtenida del paso anterior.

La asincronía permite otorgarle a la simulación una apariencia más de vida, ya que el mundo real no es sincrónico. El problema de esta es que al seleccionarse células al azar para actualizar, se desaprovecha en gran medida la caché de cada procesador, ya que constantemente se generarán cache miss dado que las células pueden no estar ubicadas cercanas en la matriz. El uso efectivo de la caché se ve beneficiado en los autómatas celulares sincrónicos, donde el orden de actualización no genera comportamientos influenciados por este orden. Cuando se utiliza asincronía, mantener un orden que permita la utilización de caché efectiva requeriría influenciar el comportamiento de las células, ya que este orden podría beneficiar a las que se ejecuten primero según este orden, dado que estas podrán moverse a donde quieren mientras que si una de las siguientes intentara moverse al mismo lugar, no podria y deberia decidir moverse a otro lugar o no moverse.

3 Soluciones secuenciales

Una posible solución secuencial sería la ejecución de la función de actualización de cada célula iterativamente, randomizado el orden de estas en cada iteración para evitar que unas tomen prioridad sobre otras. En cada iteración, cada célula Dictyostelea vera aquellas que la rodea en busca de quimicos para seguir, los cuales pueden llevar a otras células Dictyostelea que hayan perdido toda su energía o a comida. El programa consta con 2 matrices, una matriz en donde se posicionan las células y otra en donde se pondrán los valores de los químicos del mundo.

Las células vivas se moverán hacia donde perciban mas quimicos o de forma al azar en caso de no percibir ningún químico. Cada célula viva se moverá de a un paso.

Una vez una célula Dictyostelea se quede sin energía o perciba químicos de otra célula Dictyostelea que se queda sin energía liberará químicos para avisar a las demás células Dictyostelea la ausencia de comida en el entorno y comenzar la etapa de agregación de las células Dictyostelea. Esta liberación de químicos se verá reflejada en el mundo para que otras células Dictyostelea puedan percibirla, a medida que se aleja de la célula Dictyostelea , la señal química de agregación irá disminuyendo en potencia, por lo que será necesario que otras células Dictyostelea lo repliquen para poder alcanzar células Dictyostelea más alejadas.

El programa, por lo tanto, consistiría de una iteración por cada actualización que recorrerá N células vivas, examinara su entorno, se moverá acordemente a este análisis y finalmente libera una señal de agregación con un alcance M en caso de haber recibido una o de haberse quedado sin energía. Por lo tanto el programa quedaría con orden [N*M*M]. Habría que tener en cuenta también la cantidad de cache miss ocurridos en caso de que la matriz sea demasiado grande, lo cual podría aumentar el tiempo en gran medida.

4 Soluciones concurrentes

Es fácil ver que la concurrencia del programa es aplicable a las células vivas, pero hay que considerar la sincronización de estas, ya que todas acceden al mismo mundo, el cual actúa como variable compartida. De no tenerse en cuenta la sincronización se podría generar la pérdida o la multiplicación de células vivas cuando dos o más intenten moverse al mismo lugar.

Para poder aprovechar la caché, se tomará en cuenta la proximidad de las células para realizar el procesamiento, por esta razón, se optará por utilizar un acercamiento sincrónico al autómata celular. Se comienza dividiendo de forma estática las células dependiendo de la cantidad de procesadores a utilizar, ya que todas las células vivas realizan la misma cantidad de trabajo no es necesario asignarlas dinámicamente a los diferentes procesos.

En una solución con pasaje de mensajes, se utilizará una estrategia de maestro esclavo, el maestro poseerá el mundo y realizará la división de las células vivas para enviar a los esclavos para que procesen la funcion de actualizacion. Además el maestro deberá enviarle a los esclavos una copia de las secciones del mundo que los esclavos necesiten según las posiciones de las células vivas que reciban. Los esclavos realizará el procesamiento de la función de actualización y

se comunicaran con el maestro en caso de que exista una posible colisión con las células asignadas a otros esclavos. El maestro resolverá el conflicto de forma asincrónica mientras que los esclavos pueden seguir procesando otras células que no poseen conflictos. En caso de aumentar la cantidad de procesadores es posible la utilización de una configuración tipo árbol, en la que los esclavos podran funcionar como coordinadores de otros esclavos de un nivel inferior, poseyendo una porción mayor del mundo que estos y resolviendo conflictos entre los esclavos del nivel inferior, enviando los conflictos que se generen con otros coordinadores al nivel superior.

Una vez terminado el procesamiento, los esclavos enviaran al maestro la nueva porción del mundo procesada y el maestro se encargará de realizar el join de estas porciones formando la nueva iteración del mundo. Posteriormente, el maestro podrá volver a enviar células para que los esclavos procesen.

La ventaja de esta solución es su escalabilidad, ya que permite la utilización de gran cantidad de procesadores, lo que permite el procesamiento de mayor cantidad de células vivas. El problema está en que al aumentar el tamaño del mundo, las porciones que serán enviadas a cada proceso serán mayores, y las secciones de conflicto aumentarán, lo que podría generar un overhead en la copia de las secciones a los procesos y un cuello de botella en el maestro, aunque este último se podría ver reducido utilizando coordinadores de nivel inferior con la técnica antes mencionada.

Utilizando memoria compartida el proceso principal comenzará ordenando las células vivas en un arreglo. Cada proceso leerá de ese arreglo y tomará de forma estática una porción para su procesamiento, ya que el procesamiento de todas las células es el mismo. Al utilizar memoria compartida, todos los procesos tienen acceso a la matriz del mundo, por lo cual podrán leer de ella directamente sin necesidad de copias.

Se poseerá una matriz del mundo en la cual se poseen los pesos de los químicos y otra que poseerá las posiciones de las entidades en el mundo (comida y células vivas). Cada una de estas matrices tendrá una copia, de esta forma se logra crear una memoria del mundo previo al cambio, ya que de otra forma surgirían problemas con la asincronía del mundo, debido al orden del procesamiento de las células influenciado por su posición en la matriz. Las células accederán a la matriz con los pesos de los químicos para decidir dónde moverse. Debido a que nunca se realizan escrituras sobre el mundo de pesos utilizado como memoria no es necesario la utilización de ningún lock. Una vez decidido dónde moverse, las células deberán comparar sus movimientos con los de otras células para evitar colisionar, para esto se utiliza un mundo de entidades. Cuando se compare, se pedirá el lock de la posición que se desee comparar con la posición a la que se desee mover en el nuevo mundo de entidades y luego, en caso de que no haya ninguna célula en la siguiente posición, se escribirá la nueva posición de la célula en el que será el siguiente mundo de entidades. Este lock también se aprovechará para comprobar la existencia de comida en la posición a la que se moverá la célula. De esta forma se maximiza el paralelismo, ya que las células solo lockearan la celda a la que desean moverse. Para cambiar el valor en el nuevo mundo de entidades de la celda en la que se encontraban no será necesario un lock, ya que ninguna célula intentará acceder al valor de esta celda en el nuevo mundo de entidades debido a que al estar ocupado el lugar en el viejo mundo de entidades, las células no intentarán moverse a esa posición.

En caso de haber encontrado comida deberán modificarse los pesos asociados a la comida de esa posición del mundo de pesos químicos. Para esto se pedirán los locks de cada posición afectada individualmente, para evitar de esta forma crear deadlocks y maximizar el paralelismo.

En cuanto al proceso de agregación de las células,se pedirán los locks de forma individual de cada celda afectada del mundo de pesos químicos para realizar la modificación de los pesos.

Es posible que se reduzca el overhead generado por los locks del mundo de pesos químicos utilizando separando el mundo en bloques y utilizando locks sobre estos. Esto dependerá del tamaño del mundo y del alcance de las señales químicas.

5 Solución y herramientas utilizadas

Para la implementación del programa se utilizo el lenguaje cython, el cual simplifica la escritura de módulos de extensión para python en C y C++. Este lenguaje me permitió utilizar la api de C OpenMP la cual permite la programación de programas con multiprocesamiento de memoria compartida junto al sencillo manejo de matrices proporcionados por numpy en python con una sintaxis más clara y mayor velocidad.

Se eligio Cython sobre Python con multiprocessing debido a la mayor velocidad de que otorga Cython al permitir la utilización de librerías de C. Otra ventaja de Cython es que facilita el uso de memoria compartida al permitir el uso de OpenMP mientras que en multiprocessing es necesario la utilización de las clases Value y Array, lo cual requeriría reformatear la matriz del mundo para poder compartirla entre procesos, y el módulo threading de Python está limitado por el Global Interpreter Lock (GIL), el cual limita el paralelismo limitando el programa a un proceso con múltiples threads evitando que se utilicen más de un procesador.

Se utilizan 2 matrices de 2 dimensiones, world y `entity_map`: world poseerá la concentración de sustancias químicas del mundo, las células utilizan estas concentraciones para decidir hacia donde moverse, y `entity_map` posee la matriz donde están ubicadas las células y la comida.

Al comenzar el programa se inicializarán las células y la comida en sitios al azar del mundo. Por cada comida situada se situará en world su señal química correspondiente.

Las células agregadas al mundo también serán agregadas en orden a un arreglo de células, el cual también poseerá la energía disponible por cada célula. Este arreglo es el que posteriormente será dividido estáticamente entre los procesos para su procesamiento.

En cada instancia de tiempo se correrá en paralelo el procesamiento de cada célula Dictyostelea.

El procesamiento de cada célula Dictyostelea consta de los siguientes pasos:

- Exploración: se mirara cada valor vecino en el world para descubrir algún rastro químico el cual seguir, se tomará aquel rastro que posea mayor valor, priorizando los rastros químicos de las señales de agregación de otras células, para esto, las señales de agregación tendrán siempre mayor potencia que las señales de comida. Este paso no requiere bloqueo para sincronización debido a que nunca se escribirá la matriz del mundo utilizada como memoria del mundo anterior.
- Movimiento: en caso de haber descubierto algún rastro químico, se intentará ir hacia el rastro más fuerte. Se comprobará en el mundo de entidades utilizado como memoria si es que la posición a la cual se desea ir no está ocupada por otra célula. Esta comprobación no requiere sincronización ya que la matriz del mundo de entidades utilizada como memoria es de sólo lectura dentro de la iteración. En caso de que no haya célula en el objetivo, se procederá a comprobar que ninguna célula se haya movido a esa posición comparando con el valor en la matriz de entidades nueva. Luego de comprobar que el lugar seleccionado está libre de esta manera, se procede a mover la célula indicando su movimiento en el nuevo mundo de entidades. El paso de lectura y escritura del nuevo mundo de entidades se realizará contando con un lock para la celda de la matriz a la que se quiera mover, de esta forma se evita que las células que intenten moverse al mismo tiempo al mismo lugar colisionen y generen errores. Una vez realizado este movimiento, se elimina la célula de su posición anterior en el nuevo mundo de entidades. Este último paso no requiere bloqueo debido a que ninguna otra célula intentará moverse a esa posición ya que la posición antigua de la célula se encuentra en el mundo de entidades que funciona como memoria, lo que evitará que otra célula intente realizar un movimiento allí. Existen 2 situaciones posibles que restringen el movimiento de una célula a la posición seleccionada. Es posible que exista una célula en mundo de entidades utilizado como memoria o es posible que otra célula haya realizado primero un movimiento a la celda objetivo y por lo tanto el lugar este ocupado en el nuevo mundo de entidades. En el primer caso, se realizará un deslizamiento, la célula buscará una ruta alternativa hacia el lugar con mayor concentración de químicos modificando su siguiente paso a uno adyacente. En el segundo caso, se considera que las células chocaron al intentar moverse al mismo lugar, por lo que la que llegue segunda no se moverá y en el siguiente paso, de ser necesario y posible, buscará una ruta alternativa. Si se encuentra comida en la celda a la que se mueve la célula la consumirá eliminando el rastro químico de esta y aumentando la energía de la célula. Para detectar la comida se utilizara el mundo de entidades. Previo a la escritura de la célula sobre su nueva posición, se tomará el valor que se encuentre allí. Si este valor indica que allí hay comida, se iterará sobre el área de influencia química de la comida reduciendo los rastros químicos según el peso químico de la comida. Este peso se reduce a medida que se aleja del origen, esto es para que las células puedan encontrar el camino hacia la comida. La eliminación del rastro químico de la comida se realiza sobre el nuevo mundo de pesos químicos. Para evitar problemas de sincronización, existen locks para cada celda de este mundo de pesos químicos, por lo que al intentar escribir sobre una celda de este, cada celda deberá obtener el lock de la celda, realizar la escritura y luego liberar el lock. Al remover del nuevo mundo de pesos químicos el rastro de la comida y sobreescribir sobre el nuevo mundo de entidades la posición en la que se encontraba la comida, se elimina todo rastro de esta, por lo que la comida dejará de influenciar las siguientes iteraciones del programa. Si al buscar un rastro químico no se encuentra ninguno, la célula se moverá de forma al azar intentando buscar un rastro. Si ocurre una colisión al intentar moverse a un lugar al azar, una de las 2 células no se moverá ya que al no haber rastro químico, la célula no posee percepción del mundo.
- Agregación: si la célula no posee mas energia o ha recibido una señal química de agregación de otra célula Dictyostelea, se generará una señal química de agregación. La energía se reducirá en 1 en cada iteración de la célula. Cada célula comenzará con una cantidad de energía E inicializada en la creación de las células. Esta irá disminuyendo en cada iteración de la célula eventualmente llegando a quedar sin energía si no se encuentra comida. La señal química de agregación se realizará liberando químicos al mundo, los cuales toman prioridad sobre los demás químicos para que las células decidan siempre moverse hacia la agregación. Para que esto sea así, los rastros químicos de la comida tienen un dominio entre (0,1], mientras que los rastros químicos de las

señales de agregación tomarán todos los valores mayores a 1. En la posición en la que se encuentra la célula emisora de la señal, el aumento del rastro químico será igual al rango de influencia incrementado en uno. A medida que se aleja de la célula, el rastro químico se reducirá en 1 por la distancia absoluta de la celda que se modificara a la célula en su eje más alejado, de esta forma, la reducción del rastro químico se realizará en forma de anillo. Se deberán bloquear los accesos a las celdas que se modifican en el nuevo mundo de pesos químicos a medida que se modifican.

Luego de cada iteración, se reemplazan el viejo mundo de entidades y el viejo mundo de pesos químicos por los nuevos y se plotea el nuevo mundo de entidades.

Como fue mencionado anteriormente, se poseen 2 matrices de locks, una para los locks del nuevo mundo de entidades y una para los locks del nuevo mundo de pesos químicos. Esto es así debido a que utilizando estas matrices se maximiza el paralelismo de las células ya que solo bloquearán las celdas que utilicen.

6 Resultados obtenidos

Se realizaron pruebas con diferentes tamaños de matrices, cantidad de células y cantidad de procesos. Estas pruebas se realizaron sobre 50 iteraciones del mundo y con una energía inicial para cada célula de 10.

Table 1: Table with times obtained in testing

Threads	Size	Alive Cells	Cumulative time	Timer per call
6	256	4096	12.966	0.259
6	256	2048	6.503	0.130
6	128	4096	8.946	0.179
6	128	2048	4.482	0.090
6	128	1024	2.328	0.047
6	64	2048	1.860	0.037
6	64	1024	0.959	0.019
6	32	512	0.203	0.004
4	256	4096	18.595	0.372
4	256	2048	9.319	0.186
4	128	4096	11.496	0.230
4	128	2048	5.967	0.119
4	128	1024	2.968	0.059
4	64	2048	2.469	0.049
4	64	1024	1.242	0.025
4	32	512	0.251	0.005
2	256	4096	31.634	0.633
2	256	2048	15.881	0.318
2	128	4096	20.762	0.415
2	128	2048	10.398	0.208
2	128	1024	5.134	0.103
2	64	2048	5.910	0.118
2	64	1024	2.162	0.043
2	32	512	0.423	0.008
1	256	4096	61.263	1.225
1	256	2048	30.740	0.615
1	128	4096	33.114	0.662
1	128	2048	16.641	0.333
1	128	1024	8.357	0.167
1	64	2048	5.292	0.106
1	64	1024	2.657	0.053
1	32	512	0.594	0.012

En los siguientes gráficos es posible observar la reducción del tiempo de ejecución al aumentar la cantidad de procesos:

Figure 3: Reducción para 4096 células en un mundo de 256x256 celdas. La línea roja es la predicción obtenida luego de realizar regresión linear. El resultado de esta regresión da una recta con slope de - 8.663.

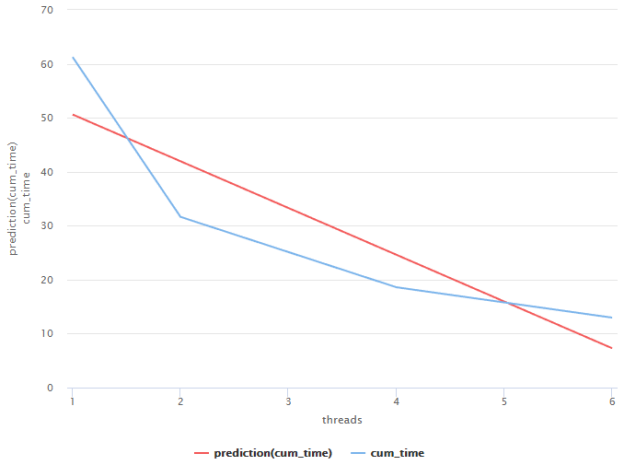
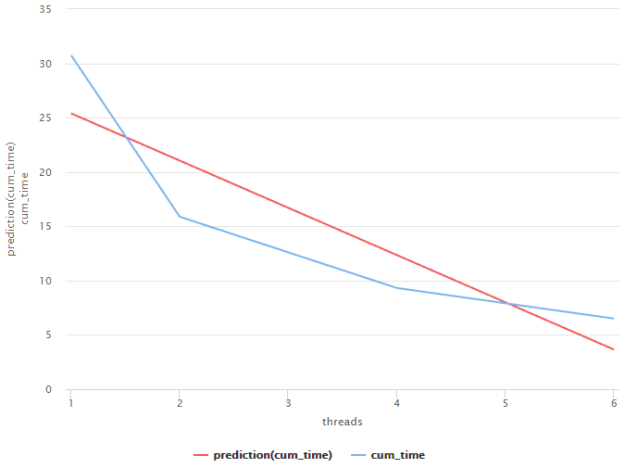


Figure 4: Reducción para 2048 células en un mundo de 256x256 celdas. La línea roja es la predicción obtenida luego de realizar regresión linear. El resultado de esta regresión da una recta con slope de - 4.349.



7 Análisis de los resultados

Para obtener la performance de la paralelización calcularé la eficiencia y el speedup del proceso. Dado que no poseo el programa secuencial más eficiente, calcularé la eficiencia y speedup con respecto a los tiempos obtenidos al ejecutar el programa con un solo procesador.

El speedup se define como el cociente del mejor tiempo secuencial sobre el tiempo de ejecución en n procesadores. Este permite conocer cuánto se aceleró la ejecución del programa al paralelizar. El objetivo es que el speedup se acerque a n ya que esto indicaría que no existe overhead en la paralelización, aunque este objetivo es muy difícil de cumplir, ya que siempre existe un overhead al paralelizar, solo se podrá alcanzar en pocos casos en raros casos.

$S_n = \frac{T_1}{T_n}$

La eficiencia se calcula como el cociente del speedup sobre los n procesadores, lo que permite obtener la cantidad de tiempo que los procesadores están realizando trabajo útil. Idealmente la eficiencia debe acercarse a 1, lo que indicaría que no existe overhead en la paralelización.

$E_n = \frac{S_n}{n}$

También es posible tener en cuenta el tiempo de ejecución de secciones de código secuencial en el código paralelizado, tal como lo mencionan la ley de Amdahl y de Gustafson. Aunque en el caso del código presentado, la sección de

Figure 5: Reducción para 1024 células en un mundo de 128x128 celdas. La línea roja es la predicción obtenida luego de realizar regresión linear. El resultado de esta regresión da una recta con slope de - 1.125.

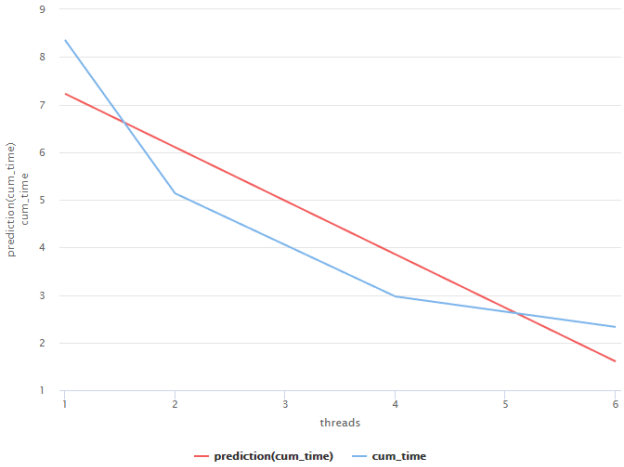
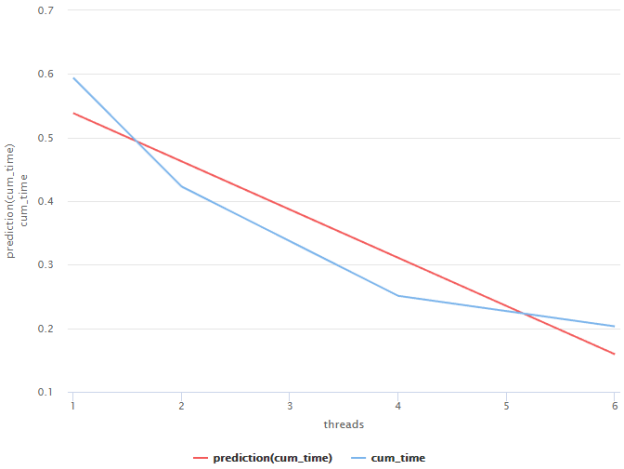


Figure 6: Reducción para 512 células en un mundo de 32x32 celdas. La línea roja es la predicción obtenida luego de realizar regresión linear. El resultado de esta regresión da una recta con slope de - 0.076.



procesamiento secuencial del código sobre el que se tomó el tiempo incluye solo la definición e inicialización de variables privadas a los procesos, cuyo tiempo es trivial, por lo que por simplicidad no se tomará en cuenta.

Observando los speedups, se puede ver que al aumentar la cantidad de células vivas y el tamaño de la matriz, el speedup y eficiencia aumentan. También se puede observar que al aumentar la cantidad de procesadores utilizados aumenta el speedup manteniendo una eficiencia aceptable. De esta forma es posible deducir que el programa es escalable, por lo que aumentar la cantidad de procesadores reducirá el tiempo del programa y aumentar el tamaño del programa otorgará mayor aprovechamiento de los procesadores con más speedup y eficiencia.

La regresión es una técnica utilizada para prediccion numerica. Es una medida estadística que intenta determinar la fuerza de una relación entre una variable dependiente y una independiente. En nuestro caso la variable dependiente será la cantidad de procesadores y la variable independiente será el tiempo de ejecución total. Utilizando regresión lineal calcule la pendiente utilizando estas dos variables, lo que otorga una aproximación respecto a cómo se comportaría el programa al aumentar la cantidad de procesadores. Una pendiente más pronunciada indica que el tiempo se reduce en mayor medida a medida que aumenta la cantidad de procesadores (la línea se calcula m*p+y siendo m la pendiente y p la cantidad de procesadores).

Con las pendientes calculadas se puede ver cómo al aumentar el tamaño del problema estas aumentan en pronunciación. Teniendo esto en cuenta se puede deducir que al aumentar el tamaño del problema se aprovechará más la utilización de más procesadores. Esto sugiere una buena escalabilidad del programa.

Table 2: Table with speedup and efficiency of some runs

Alive Cells	Size	n	T_1	T_n	Speedup	Efficiency
4096	256	2	61.263	31.634	1.93661882784	0.96830941392
4096	256	4	61.263	18.595	3.29459532132	0.82364883033
4096	256	6	61.263	12.966	4.72489588154	0.78748264692
2048	256	2	30.740	15.881	1.93564636988	0.96782318493
2048	256	4	30.740	9.319	3.29863719283	0.8246592982
2048	256	6	30.740	6.503	4.72704905428	0.78784150904
1024	128	2	8.357	5.134	1.62777561356	0.81388780677
1024	128	4	8.357	2.968	2.81570080863	0.70392520215
1024	128	6	8.357	2.328	3.5897766323	0.59829610538
512	32	2	0.594	0.423	1.40425531915	0.70212765957
512	32	4	0.594	0.251	2.36653386454	0.59163346613
512	32	6	0.594	0.203	2.92610837438	0.48768472906

Table 3: Table with slope obtained with linear regresion

Alive Cells	Size	Slope
4096	256	-8.663
2048	256	-4.349
1024	128	-1.125
512	32	-0.076

8 Conclusiones

Los autómatas celulares son objetivos ideales para la aplicación de técnicas de paralelización. Al poseer múltiples células ubicadas en la matriz realizando la misma función utilizando el valor de sus vecinos es posible el aprovechamiento de la memoria caché para la aceleración del programa aprovechando la cercanía de las células.

Dado que todas las células leerán de la misma matriz sobre diferentes lugares de esta, la matriz actúa como una variable compartida y su manejo afectará en gran medida el tiempo de ejecución del programa. Dividir la matriz para que las células puedan trabajar en paralelo sobre esta reducirá el tiempo de ejecución ya que las células permanecerán menos tiempo en espera de obtener el lock de otra célula utilizando la matriz.

El problema de la memoria compartida de la coherencia de cache que reduce el tiempo de ejecución debido a los cache miss ocasionados al intentar leer una variable compartida que se encuentra en cache y fue modificada por otro proceso no afecta demasiado al programa ya que no existen muchas situaciones en las que esto ocurra. Este problema surgirá principalmente en los sitios de conflicto, que son aquellas células de un proceso en las que alguno o algunos de sus vecinos sean compartidos por células de otro proceso.

A la hora de decidir el comportamiento de múltiples células intentando moverse al mismo lugar, se probaron los acercamientos utilizando choque y deslizamiento. Se optó por la utilización de deslizamiento cuando el sitio al que se quiere mover una célula está ocupado y choque cuando una célula se movió primero que otra a un lugar decidido, esto le otorga un comportamiento más fluido y real a las células.

Una elección importante a tomarse en cuenta fue la elección de crear un autómata celular sincrónico o asincrónico. Se optó por el autómata celular sincrónico debido a que este permite un mejor aprovechamiento de la caché, ya que para el autómata celular sincrónico el orden de la ejecución de las células no influye en el comportamiento de estas, en cambio en el asincrónico si. La capacidad de repartir las células de forma ordenada a los procesos permite que estos aprovechen mejor la caché ya que pueden aprovechar la cercanía entre las células. A pesar de que en el paralelismo no se posee un orden de ejecución fijo, al repartir las células de forma ordenadas entre los procesadores, es posible influir en su orden de ejecución, por ejemplo si un procesador es más lento que otro entonces las células de este procesador se moverán primero que las del otro.

A pesar de su simplicidad, los autómatas celulares son una forma efectiva de realizar simulaciones. Estos permiten una visualización simplificada de procesos que de otras maneras serían más difíciles de observar. En el caso de las células Dictyostelea se logró, partiendo de las especificaciones del comportamiento de estas, conseguir un comportamiento similar al observado al utilizar un microscopio sobre estas.

Esta simplicidad también puede ser problemática en caso de que el problema escale en complejidad. En caso de querer realizar simulaciones más complejas, no recomendaría la utilización de autómatas celulares dado a que estos

limitan el comportamiento del mundo. Serían necesarios demasiados cambios a la definición del autómata celular para simular sistemas más complejos. Por ejemplo si se quisiera agregar otros tipos de células al mundo con diferentes comportamientos.

References

[1] Cellular Automata. Francesco Berto, Jacopo Tagliabue https://plato.stanford.edu/entries/cellular-automata/ 19/3/2019.

[2] Cython. https://cython.readthedocs.io/en/latest/ 19/3/2019.

[3] A model for individual and collective cell movement in Dictyostelium discoideum. Eirikur Palsson* and Hans G. Othmer†‡ https://www.ncbi.nlm.nih.gov/pmc/articles/PMC27044/ 19/3/2019.

[4] Shared memory and Message passing revisited in the many-core era. Aram Santogidis CERN https://indico.cern.ch/event/450743/contributions/1116444/attachments/1220910/1784824/iCSC-Aram-final.pdf 19/3/2019.

[5] Parallel Computer Architecture: A Hardware/Software Approach. David Culler http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.4418&rep=rep1&type=pdf 19/3/2019.

[6] The Multikernel: A new OS architecture for scalable multicore systems. Andrew Baumann, Paul Barham†, Pierre-Evariste Dagand, Tim Harris†, Rebecca Isaacs,Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania †Microsoft Research, Cambridge‡ENS Cachan Bretagne https://www.sigops.org/s/conferences/sosp/2009/papers/baumann-sosp09.pdf 19/3/2019.

[7] A cellular automaton model for freeway traffic. Kai Nagel, Michael Schreckenberg https://hal.archives-ouvertes.fr/file/index/docid/246697/filename/ajp-jp1v2p2221.pdf 19/3/2019.

[8] Asynchronous cellular automata. Nazim Fatès https://hal.inria.fr/hal-01653675/document 19/3/2019.

[9] How important are updating schemes in multi-agentsystems? An illustration on a multi-turmite model. Nazim Fatès, Vincent Chevrier http://ifaamas.org/Proceedings/aamas2010/pdf/01%20Full%20Papers/11_04_FP_0210.pdf 19/3/2019.

[10] Asynchronous behavior of double-quiescent elementarycellular automata Nazim Fatès, Michel Morvan, Nicolas Schabanel, Eric Thierry https://hal.archives-ouvertes.fr/hal-00002808/document 19/3/2019.

[11] Performance and Scalability Jun Zhang Department of Computer Science - University of Kentucky https://www.cs.uky.edu/~jzhang/CS621/chapter7.pdf 19/3/2019.

[12] Rapidminer documentation https://docs.rapidminer.com/ 19/3/2019.