San Sebastián / Donostia

Simulación de propagación del SARS-CoV-2

Sistema de computo paralelo

Iñaki García
Asier Zubia

Universidad del País Vasco
Facultad de Ingeniería Informática
Ingeniería del Software

26/04/2021

Índice

1. Diseño en serie

En un primer lugar, se ha realizado la siguiente planificación en serie. La estructura del proyecto es la siguiente:

1.1 Diseño

En un principio se disponía de una planificación como se describe a continuación.

- **Código:**
  - *main.c:* fichero principal, que dispone del control de las iteraciones para llamar a su respectiva función. También se encarga de calcular las métricas.
  - *population.c:* contiene las listas de personas e inicializa y calcula todas las variables necesarias para crear la población.
  - *person_virus.c:* dispone de la función principal de crear a una persona, teniendo en cuenta todas las variables definidas en en la población (*population.c*).
  - *environment.c:* gestiona el movimiento del escenario.
  - *vaccine.c:* en el caso de que sea necesario, comprobará que se vacune a una persona según su grupo de vacunación.
- **Cabeceras:**
  - *definitions.h*
  - *environment.h*
  - *person.h*
  - *population.h*
  - *probability.h*
  - *vaccine.h*

Al compilar el programa, nos daba errores de variables repetidas, por lo que se ha optado a unificar todos los ficheros en el **main.c**.

1.2 Estructuras

1.2.1 Listas de personas

El programa dispone de **3** listas de personas:

- ***l_person_infected:*** lista de personas que están infectadas (estados 1 y 2).
- ***l_person_notinfected:*** lista de personas que no están infectadas, pero tampoco vacunadas (estado 0).
- ***l_vaccined:*** lista de personas vacunadas e inmunizadas (estado 3 y 4).

Se ha realizado esta implementación, por los siguientes argumentos:

- Cuando se realiza el cambio de estado (`void changeState(person_t *person)`), no es necesario recorrer todas las personas. Con tan solo recorrer la lista de *l_person_infected* se puede realizar la operación.
- Cuando se tiene que comprobar el radio de una persona contagiosa (`void propagate(person_t *person)`), solo se tiene que comprobar la lista de *l_person_infected*, ya que solo los infectados pueden realizar esta operación.
- En el caso de mover a las personas (`void move(person_t *person)`), se puede realizar de manera sencilla, recorriendo las listas de manera contigua. Así, se itera sobre todas las personas.
- Cuando se quiere vacunar a una persona, se añade a la lista (*l_vaccined*), para solo realizar las acciones de movimiento. También, se añaden a esta listas las personas **inmunizadas**; es decir, aquellas personas que han superado el periodo de infección. Al entender que estas personas no pueden propagar otra vez el virus, no es necesario vacunarlas, por lo que se añade a la lista automáticamente.
- Si una persona **muere**, se cambia su identificador a **-1**. De esta manera, como tenemos las listas ordenadas por su identificador, podemos recorrerlas de manera ordenada, y en el caso de que el identificador sea "-1", no se tiene en cuenta esa persona. Cuando la persona muere, también se incrementa el contador de muertes, necesario para calcular las métricas.
- Cuando una persona cambia de una lista a otra, se añade al *id+1* de esa lista, asignado dicho identificador a la persona. Es decir, cada lista tiene un identificador máximo guardado, que es incrementado cuando se añade una persona a esa lista. Por ello, las 3 listas serán como máximo el mismo valor (`size_population`), ya que en el peor de los casos se podrá llenar una lista y las otras 2 quedar vacías.

1.2.2 Escenario

La estructura del escenario se ha definido como `index_t world[][]`. El escenario es una matriz, que se crea de manera estática, para evitar problemas en las pruebas. Se podría implementar de manera dinámica de manera sencilla, como se ha hecho con el resto de estructuras.

La estructura `index_t`, se dispone para no disponer de estructuras de personas en la matriz. Dispone de dos atributos:

- `int id`: id/índice de la persona.
- `enum list l`: enumerado, que indica en cual de las listas se encuentra la persona. El enumerado dispone de los 3 valores posibles: *INFECTED*, *NOT_INFECTED* y *VACCINED*.

De esta manera:

- Se evita la duplicación de una misma instancia de la estructura; es decir, la misma persona en su correspondiente lista y en la matriz.

- No se utiliza tanta memoria, ya que la estructura es más simple.
- Cuando se quiera la estructura de la persona, se accederá a su lista mediante el atributo `enum list l` y a la posición de `int id`.
- Cuando se quieran mover las personas (`void move(person_t *person)`), no es necesario recorrer todas las posiciones de la matriz, sino recorrer las 3 listas de las personas, con su correspondientes coordenadas. Esto aumenta la eficiencia del programa, ya que con una persona se pueden realizar diferentes operaciones, sin tener que realizar múltiples bucles.

1.2.3 Persona

La estructura para almacenar las personas se llama `person_t`. Contiene los atributos mencionados en el enunciado. A destacar los siguientes campos:

- `float prob_infection`: probabilidad de 0 a 1, de que una persona sea contagiada. Después, se realiza una conversión a entero, para poder compararlo con un valor estático. Si es mayor a ese valor, se infecta.
- `int id`: identificador incremental. Coincide con su posición (índice) de la lista asociada. En el caso de que cambia de lista, se asigna el ultimo índice de la lista.
- `int state`: como se ha comentado previamente, se toman las personas recuperadas y vacunadas por igual, aunque con distinto valor del estado. Esto se debe a que cuando una persona pasa el virus, se inmuniza, dificultando así su propagación.

2. Implementación

En este apartado, se redactado las funciones más importantes del programa, para definir las reglas/restricciones que se han tenido en cuenta.

2.1 `void per_cicle()`

Método que es llamado por el método `bash`, que se encarga de ejecutar los bucles necesarios. Para ello, se inicializan las variables y se iteran por todas iteraciones del programa. Para ello, en un primer lugar se tiene en cuenta el grupo de vacunación de la iteración, y se prosigue iterando sobre las listas en un orden específico:

- En primer lugar, se leen las personas infectadas, ya que son las únicas las cuales hay que cambiarles el estado (`void change_state(person_t person)`). Es importante primero mirar el estado, ya que se pueden infectar antes a personas sanas, y así no se miraran esas personas que se han infectado. En el caso de que se cambie de estado, no se mueve.
- Después, se itera sobre los vacunados. Es necesario iterar antes sobre los vacunados que los infectados, ya que los infectados pueden pasar a este grupo, y no interesa mover a esas personas en esta iteración.
- Por último, se itera sobre las personas infectadas, comprobando si pueden ser vacunadas o no.

Para finalizar, se deberían de calcular las métricas, pero como se explicará en el siguiente Capítulo, esa tarea no se ha podido cumplir. Dicha tarea, será implementada en la parte final del proyecto. También, se imprime el estado del escenario por consola, para poder realizar un seguimiento de la ejecución.

2.2 `void change_state(person_t person)`

En el método de cambiar de estado, se tiene en cuenta el tiempo de incubación. Hasta que una persona infectada supere ese tiempo de vacunación, no podrá realizar la acción de propagar el virus. En el caso de que lo supere, se comprobará el periodo de recuperación. Durante este periodo, sí que se tiene la capacidad de propagar el virus.

2.3 `void propagate(person_t *person)`

Para comenzar, se ha definido que el radio de propagación no puede ser superior a dos. De esta manera, una persona puede propagar a 12 casillas adyacentes a la suya, teniendo en cuenta que los movimientos en diagonal consumen dos movimientos. En el caso de que pueda contagiar a una persona; es decir, que sea una persona sana y esté en su radio, se calculará la probabilidad de contagiarle.

En el caso de que se contagie, se procede a calcular la probabilidad de que muera. Esto quiere decir, que una persona solo puede morir cuando es infectada, y no se tiene en cuenta la probabilidad de su muerte durante el periodo de incubación del virus. No conllevaría mucho esfuerzo cambiar este comportamiento, ya que solo sería necesaria modificar el método de `change_state()`, teniendo en cuenta dicho periodo.

2.4 `void move(person_t *person)`

Una persona solo se puede mover, en las mismas direcciones en las que se pueden propagar el virus; es decir, en sus 12 casillas adyacentes. Para ello, se disponen de matrices (`int diagonals` e `int directions`), que disponen de todas las posibilidades a efectuar. En el caso de que la posición esté vacia, se elimina la posición anterior cambiando el identificador a "-1" y se modifica la nueva casilla del escenario.

2.5 `int vacunate(person_t person)`

El método vacunar, primero comprueba que esa persona pertenezca al grupo de vacunación. Para no complicar el ejercicio, los grupos de vacunación cambian al transcurrir un número de iteraciones. En el caso de que la persona a vacunar sea de ese grupo, se vacuna, realizando las operaciones de eliminar y añadir de las listas.

Si la persona ha sido vacunado, se devuelve **0** en el caso de que no se haya vacunado, y *1* en el caso contrario. Esto es necesario ya que las operaciones de vacunación, se realizan sobre las personas sanas. En el caso de que esa persona cambie de sano a vacunado (estado $0 \longrightarrow 4$), no se moverá en esa iteración.

3. Problemas en el desarrollo

Se han tenido diversos problemas en el desarrollo de la práctica:

- **Números aleatorios:** A la hora de sacar los números aleatorios para variables específicas, nos hemos encontrado que siempre que se ejecuta el programa, dichos valores son los mismos. Al final se optó por usar la librería *time.h* para dicho cometido. Sin embargo, seguía ocurriendo lo mismo, pero al ponerle un tiempo de espera de un segundo entre las llamadas a la función *random_number* el problema desapareció. Esto solucionó el problema pero causa otro problema más grande, el tiempo de ejecución. Ya que por cada persona se llama a dicha función dos veces, por lo que por cada persona hay que esperar 2 segundos. Si se lleva a gran escala es tremendamente ineficiente en cuanto al tiempo. Para la siguiente entrega este problema estará solucionado. Solo se ha añadido el tiempo de espera a la hora de crear las personas, para que las personas sean más diferentes en las simulaciones. En el resto de calculaciones de probabilidades, no se han añadido los tiempos de espera, por lo que el error persiste. No genera conflictos con la ejecución del programa, pero si que genera ejecuciones no tan distintas. Por ejemplo, a la hora de calcular las probabilidades de movimiento, siempre se obtienen los mismos valores, por lo que puede darse el caso de que en una ejecución las personas no se muevan ya que su movimiento sea hacia arriba, movimiento que no podrán realizar al ser inicializados en las posiciones más altas. Para solucionar esto, en las pruebas de ejecución, se han cambiado los valores de manera manual, pero se ha dejado con las probabilidades para la entrega.

- **Métricas:** En cuanto al fichero de las métricas y posiciones, no se ha podido completar de manera correcta por falta de tiempo. La función para calcular los valores necesarios para las métricas están hechas, sin embargo no funcionan correctamente (creación de ficheros). De nuevo comentar, que esta función estará bien implementada para la siguiente entrega.

- **Instancias raras en la matriz:** El mayor problema con el que se ha convivido, ha sido el problema de las instancias raras en la matriz. No se ha podido concluir con el fallo de código que hace que a la hora de imprimir el mundo, al cabo de tres o cuatro iteraciones, en la posición 1,0 de la matriz, siempre aparezca de la nada una persona no infectada. A pesar del empeño, no se ha conseguido dar caza al problema y es nuevamente un añadido más a la corrección para la siguiente entrega. Este error, junto al primero, han sido los quie más tiempo han conseguido. Sobretodo este, ya que en todo momento se ha estado comprobando el programa, sin tener éxito en la resolución del problema. Para ello, se han realizado múltiples ejecuciones con el *debugger* de C (gdb), con el cual este extraño error no ocurre. Por ende, se sospecha que este problema es ajeno o irreconocible.

- **Programa paramétrizable:** todos los lastres enumerados en esta lista, han generado que esta tarea tampoco se pudiese llevar a cabo. La idea principal, era implementar todo el programa con variables estáticas, para que en el caso de disponer tiempo al final, parametrizar el programa con un fichero. Sin embargo, a raíz de la mala gestión del tiempo causada por los errores anteriores, no ha sido posible. Por consiguiente, también queda pendiente para la sigueinte parte.

4. Diseño en paralelo

Durante el diseño en serie, se tuvo muy en cuenta el futuro diseño en paralelo. Esto conllevó a que el primer diseño tuviese que acarrear más horas, para poder realizar la menor cantidad de modificaciones en el cambio de paradigma. Todas las estructuras planteadas en el diseño en serie, deberían de sufrir la mínima cantidad de cambios.

Para llevar a cabo la implementación en paralelo, se ha definido como precondición disponer de un número mínimo de 3 nodos y el número de nodos tiene que ser impar. Estos nodos se clasificarán de la siguiente manera:

4.1 Nodo 0

El nodo inicial, 0 o P0, gestionará el escenario y las personas vacunadas. Será el único procesador que dispondrá de esta estructura, y gestionará todos los mensajes de movimientos del escenario. Gestionará las siguientes acciones:

- Recibirá mensajes de aquellas personas que han fallecido, por lo que será necesario eliminarla del escenario.
- Las personas infectadas, preguntaran por las personas que estén en su radio de contagio. Por lo tanto, P0 devolverá la lista de las casillas asociadas a ese radio, que puedan ser infectadas. Posteriormente, los nodos de personas contagiadas, deberán de preguntar por esas personas.
- En el caso de que una persona se vacune, se envía dicha persona al nodo P0. Solo tendrá que tener en cuenta, que tiene esa lista es de su posesión, por lo que a la hora de realizar los movimientos, ya dispone de esa lista.
- Imprime los mensajes por consola, para mostrar el estado de la ejecución y poder realizar una traza del programa.
- Recolecta la información de las métricas.

4.2 Nodos infectados

Los nodos infectados, son aquellos que disponen de personas infectadas. Al comienzo del programa, se reparte a los nodos infectados las personas infectadas de manera proporcional. Por ejemplo, en el caso de que se dispongan de 10 personas infectadas y 5 nodos, 1 nodo sería P0, 2 nodos serían de personas infectadas y los otros dos restantes tendrían las personas sanas. Como se disponen de 10 personas infectas, cada nodo tendría 5 personas ordenadas en orden ascendente según su identificador. Las acciones que realiza este nodo son las siguientes:

- Cambia el estado a las personas infectadas. En el caso de que mueran, notifica al nodo principal.
- Propaga el virus, con la ayuda del nodo principal. Con la lista de personas a infectar, realiza un *broadcast()* los nodos no-infectados. En el caso de que el nodo tenga a esa persona, se la devuelve. Cuando dispone de todas las personas que ha logrado contagiar, notifica a P0 de ello, para que pueda cambiar las casillas correspondientes.

4.3 Nodos no-infectados

Sigue el mismo planteamiento a los nodos infectados, a lo que al reparto de personas se refiere. Las acciones que realiza este nodo son las siguientes:

- Recibe la solicitud de un posible contagio, en el caso de que se contagio, envía a esa persona al nodo que ha realizado la solicitud.
- En el caso de que una persona sea vacuna, tendrá que modificar su lista.

4.4 Otra implementación

En un principio, entre otros diseños descartados, se planteó la posibilidad de que el escenario se situase repartida por todos los nodos. Sin embargo, esto aumentaría el tráfico de la red, empeorando el rendimiento del programa. El diseño actual carece de que puede ser el caso de que unos nodos tengan mucha carga de trabajo, mientras que otros nodos estén más "tranquilos".