Tesina de Grado
Licenciatura en Sistemas de Información

“Aplicando técnicas de programación paralela en un Prototipo de simulación para la divulgación de noticias”

Alumno

Escalante, Julián Emiliano

Directora

Dra. Piccoli, María Fabiana

Concepción del Uruguay - Entre Ríos

2020

Agradecimientos

En primer lugar quiero agradecer a mi Directora, la Dra. Fabiana Piccoli, quien con sus conocimientos, paciencia y consejos me guió a través de cada una de las etapas de este proyecto, brindándome todas herramientas necesarias para llevar a cabo el proceso de investigación.

También quiero agradecer a la Mg. Adriana Gras, por su ayuda y dedicación como docente de la cátedra.

A mi amiga Lucía, por alentarme al inicio de esta profesión y por seguir haciéndolo aún a la distancia.

A mi familia y amigos, por apoyarme aún cuando mis ánimos bajaban. En especial, quiero hacer mención de mis padres, que siempre estuvieron ahí para darme palabras de apoyo y un abrazo reconfortante para renovar energías.

A Dios, por todas sus bendiciones.

¡Muchas Gracias!

Índice

Resumen

La toma de decisiones en sistemas compuestos por múltiples componentes interrelacionadas, no es una tarea sencilla. Una posibilidad es abordarlos analíticamente o mediante ensayos, lo cual puede implicar riesgo. Otra buena solución suele ser analizarlos a través de técnicas de simulación por computadora, las cuales representan una de las herramientas más poderosas para la resolución de problemas. Su potencial es incalculable, permite abstraer sistemas del mundo real y tomar decisiones basadas en múltiples experimentos.

En el marco del proyecto de investigación "Cómputo de Altas Prestaciones aplicado en la Solución de Grandes Problemas", el desarrollo del siguiente trabajo se abordaron, en primera instancia, los conceptos básicos relacionados a los ejes de investigación: Modelos de Simulación basados en Autómatas Celulares, Computación de Alto Desempeño y Fenómenos sociales como la Difusión. Luego se propone un diseño del modelo simulación para estudiar la viralización de noticias o rumores (noticias falsas), fenómeno social de una población representada por un autómata celular (AC), donde se intenta reproducir la dinámica del comportamiento interrelacional y cotidiano en la transmisión de contenidos noticiario entre cada individuos y sus vecinos, siguiendo una serie de reglas de toma de decisiones.

Dada la naturaleza de los autómatas celulares, el diseño propuesto es llevado a una implementación computacional aplicando Técnicas de Computación de Alto Desempeño, más precisamente de Computación Paralela basada en el paradigma de Pasaje de Mensajes. Esto pretende reducir el tiempo de las simulaciones a fin de lograr resultados más rápido. El desempeño de la propuesta paralela es comparado según un análisis de métricas de desempeño, comparado con la implementación del modelo planteado en una versión secuencial tradicional.

Palabras Claves

Fenómenos de Difusión, Noticias, Autómatas celulares, Computación de Alto desempeño.

CAPÍTULO 1

Introducción

La comunicación mediada o masiva, incorporada en nuestra vida cotidiana a través de las redes sociales, programas de televisión, diarios, relaciones sociales y otros medios, nos otorga la facultad de poder producir contenido desde un origen y enviarlo a múltiples destinos simultáneamente. Esto resulta ser un aspecto conveniente para que la información verídica llegue a numerosos puntos receptores, aunque también abre paso a noticias parcialmente inciertas o directamente erróneas, enviadas con el objetivo de generar confusión en una población.

En la última década, la sociedad ha visto cómo las computadoras han tomado un importante posicionamiento en la realización de aportes y, en consecuencia, cumpliendo un rol que complementa los conocimientos de otras áreas de la ciencia para dar explicación a muchos fenómenos que nos rodean, como es el caso de la propagación de noticias o rumores a través de la comunicación online y/o presencial entre personas.

La simulación de un sistema permite diseñar un modelo lógico-matemático de la realidad reproduciendo las condiciones, comportamiento operacional y dinámico, para estudiarlo y probarlo, con el objetivo de lograr un mayor grado de conocimiento en la toma de decisiones.

La simulación mediante Autómata Celular (AC) es una de las técnicas computacionales más utilizadas en la actualidad para innovar y estudiar comportamientos de varios fenómenos, por ejemplo la difusión de enfermedades, de virus informáticos o de noticias, la propagación del fuego o la onda verde en los semáforos entre otros. Estos escenarios están compuestos por un grupo de individuos/objetos interrelacionados entre sí, cuyo comportamiento sigue algunas reglas. Por su naturaleza y características, los AC permiten aplicar técnicas de computación de alto desempeño (HPC, High Performance Computing) a fin de aprovechar el potencial computacional brindado por las computadoras contemporáneas.

En base a lo expresado anteriormente y, teniendo en cuenta la gran demanda de recursos computacionales necesarios para analizar el comportamiento de la difusión, propagación y/o dispersión de cualquier tipo de noticias: verdaderas o falsas, y si existe alguna diferencia en la modalidad de difusión de ambos tipos, se propone en este trabajo el diseño e implementación de un prototipo paralelo para simular la dispersión de noticias basado en AC.

1.1. Planteo del Problema

La propuesta presentada en este trabajo recurre a dos ejes de investigación complementarios al momento del desarrollo y experimentación, en primer lugar, proponer un diseño enfocado en modelos de simulación por medio de autómatas celulares imitando la interrelación e intercambio de información de nodos desde una perspectiva de fenómeno social. El segundo eje se basa en implementar dicho diseño aplicando técnicas de computación de altas prestaciones con el fin de poder aportar una herramienta que nos permita recrear y estudiar, bajo condiciones o reglas de influencia, la transmisión de noticias, con distintos grados de veracidad, en una población.

1.2. Hipótesis

- H1: Las noticias falsas se dispersan a mayor velocidad que las verdaderas.

1.3. Objetivos de Investigación

En esta sección expresamos el objetivo general y los específicos del trabajo planteado, a fin de validar o refutar la hipótesis planteada.

- **Objetivo General:**

  Diseñar e implementar un prototipo paralelo para simular un modelo de dispersión de noticias basado en autómatas celulares. A través de este desarrollo, se pretende analizar el comportamiento de la difusión, propagación y/o dispersión de cualquier tipo de noticias: verdaderas o falsas, y si existe alguna diferencia en la modalidad de difusión de ambos tipos de noticias.

- **Objetivos específicos:**

  1. Relevamiento bibliográfico.
  2. Análisis de distintos modelos de dispersión, tanto de información como de transmisión infecciosas, entre otros.
  3. Determinación de los posibles estados en los que pueden estar los individuos y las reglas de cambio de estado.
  4. Diseñar un Autómata Celular considerando los estados y las reglas determinados, además de la inclusión de técnicas de Computación de Alto Desempeño.
  5. Implementar el modelo diseñado considerando herramientas estándares de desarrollo de Computación Paralela para lenguaje Python.
  6. Evaluar y analizar el desempeño de la solución alcanzada.

1.4. Resumen de los Capítulos

Capítulo 2: Marco Conceptual

Las nuevas tecnologías y la demanda de mayor poder de cómputo hacen que se busquen alternativas a la computación secuencial para resolver grandes problemas. En este capítulo se desarrollan los dos conceptos básicos involucrados en la tesina: la Computación de Alto Desempeño (HPC) y los Autómatas Celulares (AC). En el primer caso se hace énfasis en los Sistemas Paralelos, sus características y técnicas, mientras que para AC se explican sus fundamentos e importancia en la simulación de Sistemas Complejos, observando la relación existente entre ambos aspectos.

Capítulo 3: Casos de Estudio y Resultados Experimentales

En la primera parte del capítulo se presentan los antecedentes bibliográficos consultados en referencia a la temática de AC, HPC y modelos propuestos para la simulación de divulgación de noticias/rumores.

Seguido a esto, continuando con esta investigación, la explicación de la Teoría del Juego de la vida, base para aspectos de diseño y desarrollo del modelo HPC-AC_{N-R}, el cual fue diseñado e implementado como un prototipo paralelo. Finalmente, se presentan resultados experimentales sobre un ambiente específico, en el cual se tomaron los tiempos en ejecución para evaluar parámetros de desempeño como la aceleración y eficiencia alcanzada por prototipo. Además se consideraron otros aspectos de estudio en relación al fenómeno de difusión, como por ejemplo ¿Qué noticia se transmite más rápido, la verdadera o la falsa? ¿Influye la densidad poblacional en la difusión de noticias?, entre otros.

Capítulo 4: Conclusiones y Trabajos Futuros

En este capítulo se plantean las conclusiones pertinentes en base a las apreciaciones de los resultados obtenidos y se plantean algunas líneas de trabajo futuro a fin de mejorar el modelo HPC-AC_{N-R}, ampliar sus prestaciones, validando sus resultados. Además se pretende ampliar la aplicación de otras tecnologías de HPC en el modelo de estudio para alcanzar mejor desempeño.

CAPÍTULO 2

Marco Conceptual

Como se mencionó en el capítulo anterior, las nuevas tecnologías y la demanda de mayor poder de cómputo hacen que se busquen otras alternativas a la computación secuencial para resolver grandes y/o complejos problemas. En este capítulo vamos a desarrollar los dos conceptos básicos involucrados en esta tesina: la Computación de Alto Desempeño (HPC) y los Autómatas Celulares (AC). En el primer caso nos enfocamos en los Sistemas Paralelos, sus características y técnicas, mientras que para CA detallamos sus fundamentos e importancia en la simulación de Sistemas Complejos, observando la relación existente entre ambos aspectos.

2.1. Computación de Altas Prestaciones

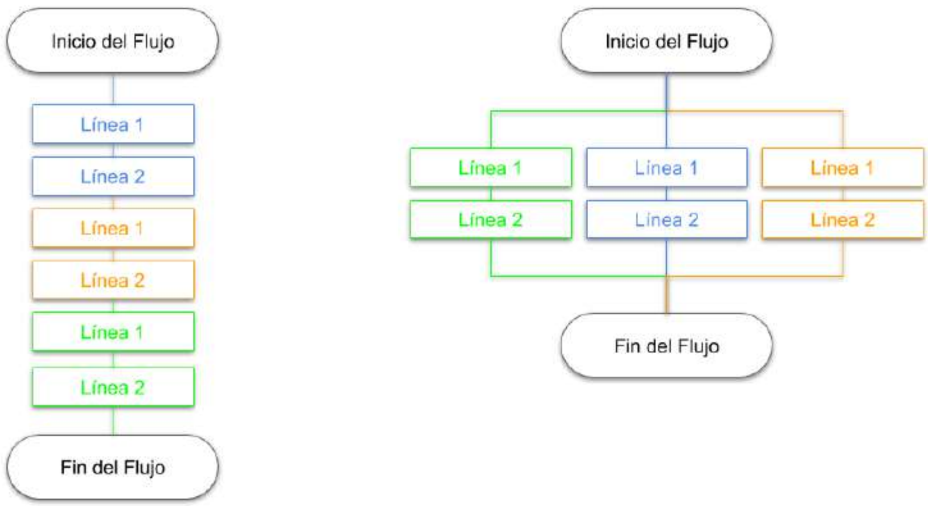
Como se expresa en Piccoli (2011), la comunidad académica y científica ha visto y ocupado un papel importante en la determinación de los grandes avances en el hardware de las computadoras y las comunicaciones, hoy en día todas las computadoras cuentan con más de dos procesadores y, generalmente, están conectadas a otras computadoras o dispositivos. El Software, acompañó el progreso brindándonos metodologías y técnicas para el desarrollo de nuevos sistemas, los cuales permiten considerar estas nuevas arquitecturas logrando aplicaciones con mejor desempeño. Ejemplo de ellos son los Sistemas Distribuidos y los Sistemas Paralelos. En este trabajo nos enfocamos en estos últimos.

2.1.1. Computación y Sistemas Paralelos

Para Almeida, Giménez, Mantas y Vidal (2008), estamos en presencia de computación paralela (CP) cuando utilizamos a nuestro beneficio el grado de simultaneidad de las tareas, es decir, existen tareas que pueden realizarse al mismo tiempo. Cabe destacar, que si bien está relacionado con los sistemas paralelos (SP), son conceptos distintos, CP hace referencia a arquitecturas con múltiples unidades de procesamiento, capaces de soportar procesamiento paralelo, mientras que los SP se caracterizan por el procesamiento y manipulación concurrente de la información perteneciente a uno o más procesos, quienes trabajan conjuntamente en la resolución del mismo problema. Requieren tanto simultaneidad espacial como temporal en la ejecución de eventos relacionados e involucrados en la resolución de un único problema (Piccoli, 2011).

Los SP ejecutan distintas instrucciones en mismo instante de tiempo, brindándonos una herramienta poderosa para obtener un mejor desempeño de la arquitectura sobre la que se está trabajando. Existen paradigmas de programación paralela, los cuales establecen aspectos conceptuales relacionados a la metodología de diseñar algoritmos paralelos. La Figura 2.1 muestra un ejemplo de cómo un código secuencial (Figura 2.1 (a)) puede ser planteado según una solución paralela (Figura 2.1 (b)).

Si bien la programación paralela nos permitirá resolver problemas en menor tiempo, no todos los problemas son adecuados para ser resueltos en paralelos. Existen algunos que exigen un orden obligatorio e inflexible en sus operaciones, lo cual hace imposible la aplicación de esta nueva metodología.



(a) Código Secuencia **(b) Código Paralelo**

Figura 2.1. Programación Secuencial vs Programación Paralela

Fuente: Figura 2.1

En la elaboración de un algoritmo paralelo es vital reconocer si el problema cumple con el concepto de Paralelismo ilustrado en la Figura 2.1 (b). Por ejemplo, imaginemos que un sujeto tiene que realizar dos tareas en el día:

a) Llegar de su casa al trabajo.

b) Hacer una presentación y exponerla a su jefe.

El problema puede ser resuelto de dos maneras, ellas son:

- *Secuencialmente:* Puede ir al trabajo y a su llegada, empezar a trabajar en la presentación para luego exponerla.

- *En paralelo:* En este caso, junto a un compañero de equipo coordinan las tareas a realizar y mientras viaja a la oficina, su compañero trabaja en la presentación. Al llegar, el compañero le entrega la presentación y él se dispone a realizar la exposición a su jefe. En este caso, el trabajo fue realizado por dos personas

personas, con acciones simultáneas en el tiempo: mientras uno viajaba, el otro preparaba la presentación.

En otras palabras, “paralelismo es la posibilidad de dividir de un determinado problema computacional en partes y resolverlas en forma independiente” (Almeida et al, 2008).

En relación a CP, se desprende la existencia en todo sistema paralelo de dos aspectos bien distintivos: el Hardware y el Software. Cuando hacemos referencia al Hardware hablamos puntualmente de la arquitectura, en este caso, capaz de soportar procesamiento paralelo, contando con varias unidades de procesamiento dispuestas de alguna manera, dando lugar a distintas arquitecturas según la configuración de las conexiones de los procesadores y la memoria.

Una de las clasificaciones más conocidas del hardware es la “taxonomía de Flynn”, la cual considera el flujo de instrucciones y de datos a administrar simultáneamente (Flynn, 1995) (Pacheco, 2011). En la Figura 2.2 se muestra la clasificación y las cuatro categorías resultantes.

Las características de cada una de las categorías son:

- SISD: Un sistema con un único flujo de instrucciones y un único flujo de datos. Estas arquitecturas ejecutan una sóla instrucción sobre un elemento de datos a la vez. Ejemplo de ella es la arquitectura von Neumann clásica.
- SIMD: En esta categoría existen diferentes unidades de procesamiento supervisadas por una única unidad de control. Todos los elementos de procesamiento reciben, desde la unidad de controla misma instrucción a ejecutar sobre diferentes datos. Existe un único contador de programa y el sincronismo es la característica principal.

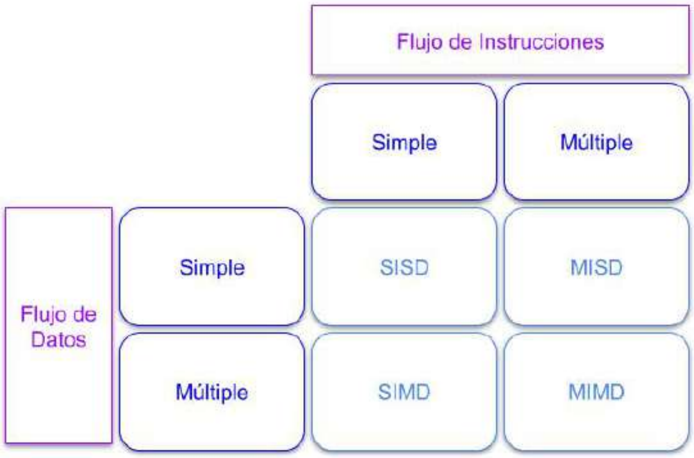


Figura 2.2. Taxonomía de Flynn.

Fuente: Ortega, J; Anguita, M; Prieto, A. (2005). Arquitecturas de Computadores. Madrid, España. International Thomson Editores Spain. Página 26.

- MISD: Las *n* unidades de procesamiento reciben diferentes instrucciones, las cuales operan sobre el mismo dato. No existe ninguna arquitectura real que implemente este tipo de computadoras.
- MIMD: Como sugiere su nombre, son los sistemas de múltiple instrucción sobre múltiples datos: ejecutar múltiples instrucciones independientes, cada uno de los cuales sobre su propio flujo de datos. Los sistemas MIMD son colecciones de procesadores autónomos que pueden ejecutarse a su propio ritmo. Los MIMD pueden ser sistemas de memoria compartida o de memoria distribuida. En los sistemas de memoria compartida, todos los procesadores o núcleos pueden acceder directamente a cada ubicación de la memoria, la comparten; mientras que en los sistemas de memoria distribuida, cada procesador tiene su propia memoria.

También es posible caracterizar las arquitecturas como “fuertemente acoplada” o “débilmente acoplada”. En la primera categoría se encuentran aquellas computadoras donde el grado de interacción entre los procesadores es alto. En contrapartida, son “débilmente acopladas” si la interacción es poco frecuente. La mayoría de las computadoras MIMD comerciales están dentro de esta última categoría.

2.1.2. Paradigmas Paralelos según el Objeto de División

Un paradigma es todo aquel modelo que describe estructuras o patrones típicos para resolver un problema. Por ejemplo, un paradigma muy conocido, es el de resolución de problemas “*divide y vencerás*”, quien se caracteriza por resolver un problema a través de la solución de subproblemas más pequeños y con la misma naturaleza que el problema original. Para los SP existen distintos paradigmas, los cuales dependen de qué se paraleliza del problema, si los datos o las tareas. Así surgen dos tipos paradigmas, cada uno tiene las siguientes características:

- *Paralelismo de datos*: Se caracteriza por la ejecución paralela de la misma operación sobre distintos datos. En otras palabras, múltiples unidades funcionales aplican simultáneamente la misma operación a un subconjunto de elementos. Dicho subconjunto es una partición del conjunto total de datos.

  Este paradigma está orientado a arquitecturas con un simple espacio de direcciones o a aquellas con memoria distribuida físicamente. En este último caso, la distribución de los datos juega un papel importante y el equilibrio de la carga también (Grama, Gupta, Karypis, Kumar, 2003).

  Aplicar paralelismo de datos tiene numerosas ventajas, una es la simplicidad de programación. La implementación final es un programa con un único thread de control. Generalmente un programa paralelo es una secuencia de pasos, algunos de ellos paralelos y otros secuenciales.

  Como dificultad, se puede mencionar que la división de los datos, en la mayoría de las herramientas lenguajes, debe ser especificada por el programador, quien generalmente decide cómo realizarla.

  Dadas las características de los problemas resueltos con paralelismo de datos, estos son aptos para resolverse en computadoras SIMD.

- Paralelismo de tareas o control: El Paralelismo de Control consiste en aplicar simultáneamente diferentes operaciones a distintos elementos de datos. El flujo de datos entre los distintos procesos que aplican paralelismo de control puede ser muy complejo.

  El paralelismo de control, si bien no es simple de programar, su principal característica es la eficiencia y adaptabilidad para resolver problemas con estructuras de datos irregulares.

Una buena herramienta para entender este paradigma es desarrollar el grafo de dependencias de las tareas involucradas en SP. En él se reflejan las interrelaciones de las tareas y sus dependencias. Por ejemplo si el grafo describe una camino simple y lineal entre las distintas tareas, entonces se dice que la solución es un pipeline.

Una computación pipeline es un caso especial del paralelismo de control donde la computación se divide en etapas; cada etapa trabaja en paralelo sobre una parte del problema tomando como entrada los datos producidos por la etapa anterior y dando como salida la entrada de la etapa siguiente. En este flujo de datos el primer y último estado del pipe son la excepción, la entrada del primero y la salida del último son, respectivamente, los datos de entrada y de salida de la solución planteada.

Los algoritmos que aplican paralelismo de control se adaptan mejor a las arquitecturas MIMD. Sin embargo, esto no siempre ocurre.

Al paralelismo de datos también se lo denomina modelo SIMD, aunque muchas veces se lo generaliza como modelo SPMD, Simple Programa - Múltiples Datos. En este caso la misma secuencia de múltiples operaciones, es aplicada a múltiples datos, el sincronismo tiene lugar al inicio y al finalizar la secuencia de operaciones.

Varias generalizaciones para el modelo de datos paralelos fueron propuestas. Ellas permiten el anidamiento de constructores paralelos de datos para especificar computaciones paralelas a lo largo del anidamiento y sobre estructuras de datos irregulares. Estas extensiones incluyen la capacidad de invocaciones paralelas anidadas, es decir combinan la facilidad de programación del modelo de paralelismo de datos y la eficiencia del modelo de paralelismo de control. Si los dominios de datos son en sí mismos estructurados, y si las operaciones paralelas a aplicarles también lo son, se está en presencia de un nuevo modelo de paralelismo: *Paralelismo Anidado de Datos*.

Los problemas resueltos a través del paradigma *Divide y Vencerás* son los que mejor se adaptan para ser resueltos aplicando paralelismo anidado de datos. La metodología propuesta por esta técnica consiste en dividir el problema en múltiples subproblemas, los cuales son resueltos independientemente y las soluciones de cada uno son combinadas en una única solución. Cada subproblema es resuelto de la misma forma que el problema original, pero en menor escala. De esto se deduce que la técnica procede en forma recursiva hasta que el problema no pueda ser dividido más.

Los algoritmos del tipo “divide y vencerás” brindan la oportunidad de explotar el paralelismo no sólo a nivel de tareas sino también a nivel de datos en las fases de división y combinación. El paralelismo de datos se introduce, entonces, haciendo que cada procesador trabaje en una subsección de los datos de entrada en la fase de división y en una subsección de los datos de salida en la fase de combinación. El paralelismo de control está presente en las invocaciones paralelas recursivas resultantes de la división del problema (Piccoli y Printista, 2001).

2.1.3. Paradigmas Orientados a la Arquitectura

Si extendemos el espectro de paradigmas paralelos, haciendo hincapié ya no en las tareas y/o datos, si en la memoria y en los procesadores, determinando la topología de la conexión entre los procesadores y los módulos de memoria, estos son únicos o propios, es decir, si todos los módulos integran un único espacio de direcciones: *Memoria Compartida*, o si se considera propios de cada procesador: *Memoria Distribuida*.

Las características principales de cada uno son:

- *Paradigma de Memoria Compartida:* Bajo este modelo, la memoria se caracteriza por ser pública y, como lo ilustra la Figura 2.3, accedida por todos los procesos (Piccoli, 2011)(Almeida et al, 2008)(Pacheco, 2011). Los autores consultados coinciden en que aquí toma protagonismo el concepto de hilo (thread en inglés): Un hilo en ejecución puede ser definido como un flujo de instrucciones independientes que permite ser planificado para su ejecución como tal por el sistema operativo (Almeida et al, 2008).

  Desde la perspectiva del programador, la memoria compartida trae consigo una atracción especial producto de la conveniencia de compartir los datos. Por lo tanto, un enfoque obvio para el problema de coordinar el trabajo de los núcleos es especificar que ciertas ubicaciones de memoria están "compartidas". Este es un enfoque muy natural para la programación paralela. Sin embargo, se presentan criterios para prevenir futuros problemas en la programación de sistemas de memoria compartida relacionados al hardware como la gestión de los accesos a las direcciones de la memoria, implementando mecanismos para proveer la consistencia de la misma (Piccoli, 2011)(Pacheco, 2011).

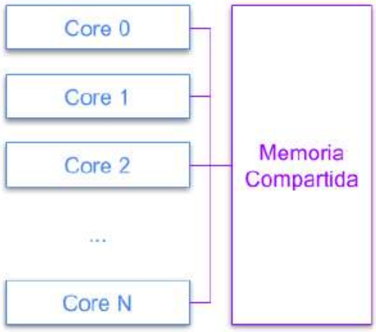


Figura 2.3. Arquitectura de Memoria Compartida.
Fuente: Pacheco, Peter S. (2011). An Introduction to Parallel Programming. Burlington, USA. Elsevier Inc. Página 9.

- *Paradigma de Memoria Distribuida o Pasaje de Mensaje:* A diferencia del modelo anterior, este paradigma es más antiguo y mayor utilizado en la programación de máquina paralelas (Figura 2.4.), esta última característica está ligada a su surgimiento.

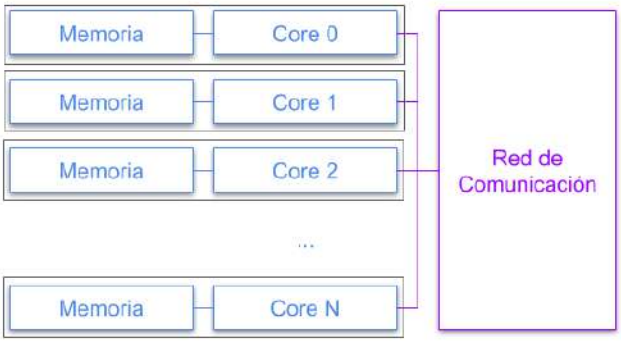


Figura 2.4. Arquitectura de Memoria Distribuida
Fuente: Pacheco, Peter S. (2011). An Introduction to Parallel Programming. Burlington, USA. Elsevier Inc. Página 9.

Normalmente el diseño del programa paralelo comprende varias tareas que solucionan un subproblema del problema mayor. A cada tarea se le asignan recursos del sistema, incluyendo la memoria, la cual es propia. En este modelo, se habla de procesos y no de threads.

Los procesos se comunican mediante el envío y recepción de mensajes. La transferencia de mensajes requiere operaciones cooperativas por parte de los procesos, a una operación de envío le corresponde con una operación de recepción (Almeida et al, 2008).

Sin embargo este modelo no resulta ser tan atractivo como de memoria compartida para la tarea de programación, demanda la especificación explícita del pasaje de mensaje en el código. A pesar de las dificultades de programación, tiene sus ventajas, una de ellas es no necesitar de mecanismo de control por acceso simultáneo a datos, lo cual mejora la performance (Piccoli, 2011). Existen herramientas que facilitan la programación paralela con pasaje de mensajes, algunas definidas como estándares, entre ellos se destaca MPI (Interfaz de Paso de Mensajes) con la cual trabajaremos a lo largo del proyecto. En secciones posteriores se detallan las características de MPI.

En un SP encontraremos una combinación de los paradigmas antes mencionados: orientados al objeto a dividir y a la arquitectura. Es decir, podemos encontrar soluciones paralelas que aplican paradigma de datos en memoria distribuida por ejemplo.

2.1.3.1. Herramientas para Pasaje de Mensajes: MPI

El modelo de paso de mensajes generalmente se aplica en los casos cuando los procesos tienen su propia memoria, es decir estamos en presencia de una arquitectura de memoria distribuida. El programador es responsable de determinar el paralelismo y el intercambio de datos que ocurre a través de los mensajes. La implementación de este modelo de programación paralela requiere el uso de herramientas de software para ser utilizadas. Se propusieron numerosas implementaciones del modelo de pasaje de mensajes, a mediados de los 90 se definió un estándar *“de facto”* llamado MPI (sigla proveniente de su nombre en inglés: Message Passing Interface). MPI está diseñado claramente para arquitecturas con memoria distribuida, pero al ser un modelo de programación paralela es múltiple plataforma, también se puede utilizar en arquitecturas de memoria compartida (Zaccone, 2015).

MPI fue desarrollado por MPI Forum[1] y define una biblioteca estándar en la que incorpora tanto la sintaxis como la semántica de un conjunto principal de rutinas para el paso de mensajes utilizado posteriormente para el desarrollar programas portables .

Según la clasificación de Flynn, el modelo de programación que subyace en MPI es MIMD; aunque es posible establecer una adecuación mejor con SPMD, caso particular de MIMD en el que todos los procesos ejecutan el mismo programa, aunque no necesariamente la misma instrucción al mismo tiempo.

MPI se apoya en dos operaciones de comunicación básicas, send y receive (enviar y recibir) que le permite establecer una comunicación punto-a-punto entre dos procesos, para la transferencia e intercambio de información. También existen las operaciones colectivas que implica compartir datos entre más de dos procesos.

En sus primeros años, la biblioteca complementaba lenguajes como C y Fortran y, en la actualidad, se ha extendido a otros lenguajes más modernos como Python, Java, .Net, entre otros (Alonso,1997)(Message Passing Interface Forum, 2015).

Para este trabajo se utilizó la biblioteca MPI4PY (versión 3.0.3) y el lenguaje de programación Python (versión Python 3.6.9). Más detalles de las características, funciones y programación con MPI están explicitadas en el Apéndice A.

2.1.4. Etapas de diseño de un Programa Paralelo

Desarrollar Sistemas Paralelos no es una tarea fácil y, al igual que en la programación secuencial, es necesario seguir metodologías que guíen y converjan en una codificación eficiente. El diseño de algoritmos paralelos se basa en una serie de etapas necesarias para que la solución paralela realice el trabajo correctamente sin producir resultados parciales o erróneos. La Figura 2.5. esquematiza el proceso de diseño de un SP.

[1] Es el foro de normalización para la Interfaz de paso de mensajes (MPI) donde se puede encontrar documentos del estándar.

Figura 2.5. Etapa para la Resolución en Paralelo de un Problema.

Las etapas a llevarse a cabo para una correcta paralelización de un algoritmo son:

1. *Descomposición de tareas:* En esta primera fase, el problema debe dividirse en tareas que luego darán lugar a procesos independientes ejecutándose en diferentes procesos. Para hacer esta subdivisión, existen dos métodos de descomposición, el primero consiste en la “descomposición del dominio” donde se particiona el conjunto de datos (preferentemente en piezas igual tamaño) obteniendo ciertos subdominio; luego se asocia la gestión de cada partición a una tarea que realizará un conjunto de operaciones a aplicar sobre dicha partición. Esta metodología se utiliza cuando tenemos una gran cantidad de datos para procesar (Almeida et al., 2008)(Zaccone, 2015) .

   La Figura 2.6. ilustra el concepto de descomposición de datos aplicado al ejemplo de una matriz de 2x2 multiplicada por un valor escalar, Tomamos el ejemplo de Palach (2014), quien nos explica cómo paralelizar el problema de multiplicar una matriz de 2x2 (vector bidimensional), a la que llama *Matriz A*, por un valor escalar, 4 en este caso. En secuencial, se realiza cada operación de multiplicación una tras otra, generando el resultado al final de la secuencia de instrucciones. Dependiendo del tamaño de la *Matriz A*, la solución secuencial del problema puede tornarse lenta y llevar tiempo. Sin embargo, cuando se aplica la descomposición de dominio o de datos, podemos imaginar un escenario en el que la *Matriz A* se particiona, cada partición es asignada a un proceso, quien trabaja en paralelo a los otros procesos, produciendo uno de los resultados.

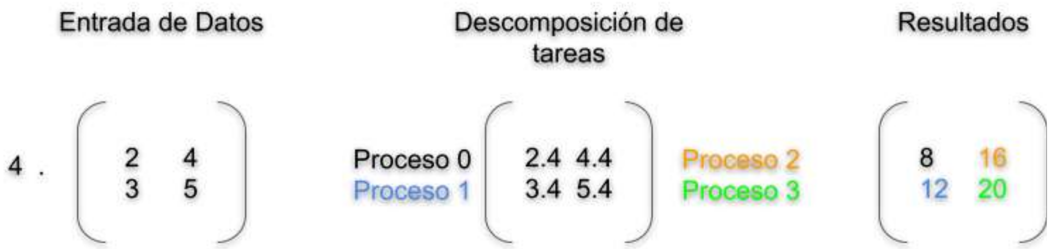


Figura 2.6. Descomposición de datos aplicado en una matriz de 2x2 multiplicada por un valor escalar.
Fuente: Palach, Jan (2014). Parallel Programming with Python. Packt Publishing. Birmingham, Reino Unido. Página 20.

El segundo método de división es la "descomposición funcional". Aquí se propone dividir el problema en tareas, donde cada una realiza una operación particular sobre los datos disponibles, todos o una parte de ellos. Almeira et al. (2008) refieren a este método como el más intuitivo y lógico. La aplicación metodológica tiene lugar en casos donde la solución del problema puede abordarse en fases bien diferenciadas.

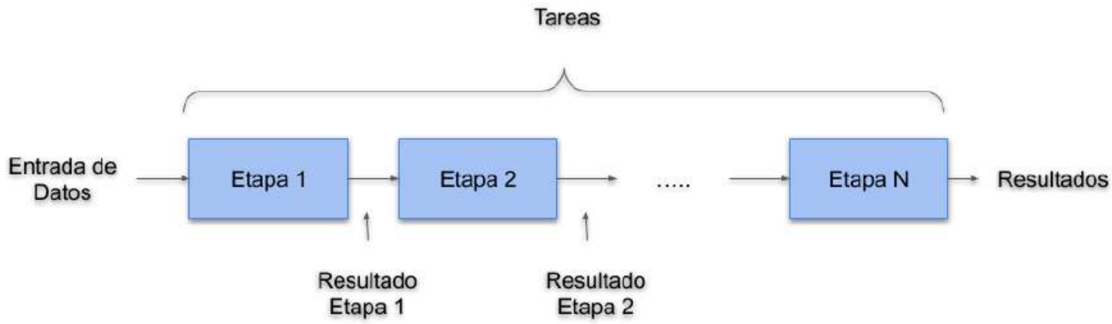
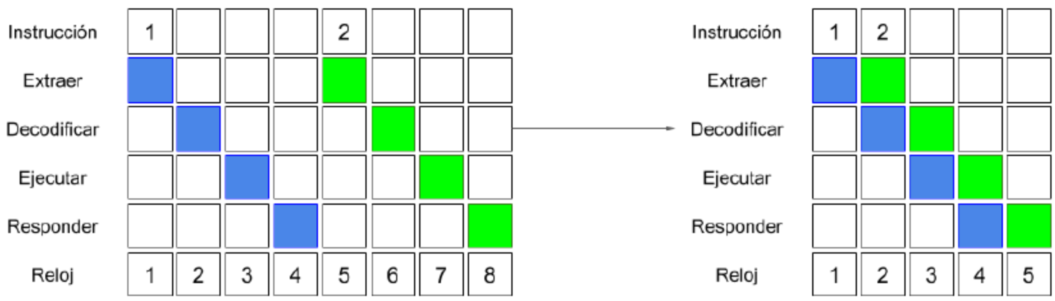


Figura 2.7. Descomposición Funcional: Estructura Pipeline.

Un claro ejemplo de este método es Pipeline (Palach, 2014), dividiendo las tareas grandes en tareas independientes más pequeñas, las cuales se ejecutan de manera paralela, donde la salida de una es la entrada de la siguiente. El modelo de pipeline podría compararse con una línea de ensamblaje en una fábrica de vehículos donde el chasis es la materia prima de la entrada. A medida que la materia prima pasa por diferentes etapas de producción, varios procesadores realizan diferentes acciones hasta el final de la producción. La Figura 2.8. muestra la comparación de la ejecución de dos secuencias de instrucciones 1 y 2 en secuencial (Figura 2.8.a.) y como un pipeline de 4 etapas: E*xtraer, Decodificar, Ejecutar, Responder* (Figura 2.8.b.). Observe que para el caso de la ejecución primero de la secuencia 1 y luego la 2, tarda 8 unidades de tiempo, mientras que si lo hace en el pipeline, demora 5 unidades.

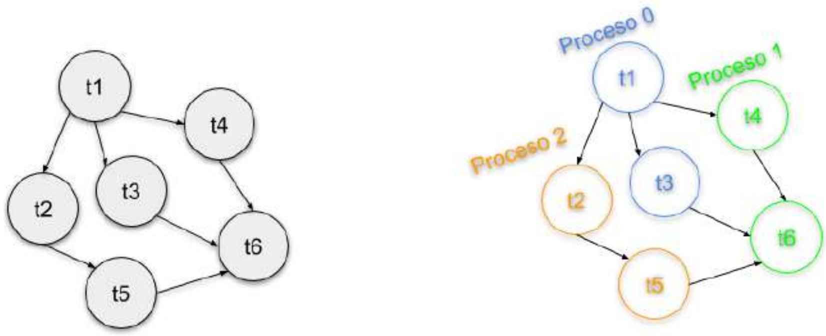


(a). Ejecución en Secuencia **(b).** Ejecución en Pipeline

Figura 2.8. Ejemplo de Ejecución en una Estructura Pipeline.

2. *Asignación de tareas:* Se especifica el mecanismo por el cual las tareas se distribuyen a los procesos. Esta fase es muy importante porque establece la distribución de la carga de trabajo. El equilibrio de carga es crucial aquí, los procesos deben funcionar con la mayor continuidad posible, evitando tiempos ociosos. Además, para una mayor eficiencia de la paralelización, es necesario limitar la comunicación tanto como sea posible entre los procesos, a menudo son la fuente de ralentizaciones y consumo de recursos (Zaccone, 2015).

La Figura 2.9.a. presenta un grafo de dependencia de las tareas, donde se puede observar cada tarea y su orden en relación a la dependencia con las demás. En la Figura 2.9.b. se muestra cómo se realizó la asignación de tareas. Puede observarse que el *Proceso 0* se encarga de las tareas *t1* y *t3*, el *Proceso 1* de *t4* y *t6* y el *Proceso 2* de *t2* y *t5*.



(a) Grafo de dependencia. **(b)** Asignación de tareas.

Figura 2.9. Grafo de Dependencia y de Asignación de Tareas.
Fuente: Almeida, Francisco; Giménez Domingo; Mantas, José M.; Vidal, Antonio M. (2008). Introducción a la Programación paralela. Madrid, España. Paraninfo. Página 270

3. *Aglomeración y Mapeo:* Aglomeración o Agrupación es el proceso de combinar tareas más pequeñas con tareas más grandes para mejorar el rendimiento. Si las dos etapas anteriores del proceso de diseño dividieron el problema en una serie de tareas que exceden en gran medida la cantidad de procesadores disponibles, y si la computadora no está diseñada específicamente para manejar una gran cantidad de pequeñas tareas, entonces el diseño puede resultar altamente ineficiente. Comúnmente, esto se debe a que las tareas se comunican entre sí. La mayoría de las comunicaciones tiene costo, el cual depende no sólo de la cantidad de datos transferidos, sino que también del costo fijo establecido por cada operación de comunicación. Si las tareas son muy pequeñas, este costo fijo puede hacer que el diseño sea ineficiente. Es por eso que la etapa de agrupación propone juntar varias tareas en una tarea más grande, a fin de minimizar los costos de comunicación.
En la etapa de mapeo, se especifica dónde se ejecutará cada tarea. El objetivo es minimizar el tiempo total de ejecución. Existen dos estrategias principales, ellas son:
    - Las tareas que se comunican con frecuencia deben colocarse en el mismo proceso para aumentar la localidad.
    - Las tareas que se pueden ejecutar simultáneamente deben colocarse en diferentes procesadores para mejorar la concurrencia.

Como puede observarse, estas dos estrategias pueden estar en conflicto, por lo que es necesario realizar una análisis de cuál priorizar con los costos y beneficios (Zaccone, 2015).

En la Figura 2.10 gráfica el proceso de plantear las tareas involucradas en la solución de un problema (partición), cómo se relacionan entre cada una de ellas (comunicación), la forma en que se pueden agrupar (aglomeración) y su posterior asignación a un proceso.

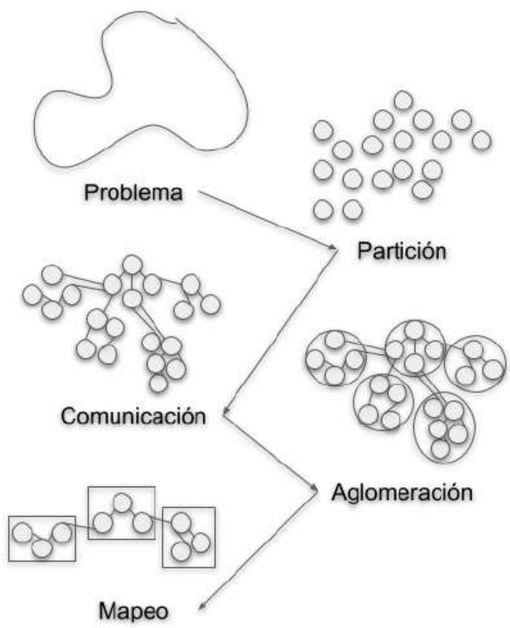


Figura 2.10. Proceso de Resolución en Paralelo de un Problema.
Fuente: [Figura 2.10]

En este trabajo se aplicó el proceso de diseño, el problema se descompuso según la Descomposición de Dominio, considerando un equilibrio en la carga de trabajo para cada tarea, las cuales se asignan a los distintos procesos, no se produce el agrupamiento de tareas. La cantidad de procesos se establece según los recursos de la arquitectura subyacente.

2.1.5. Evaluación de Sistema Paralelo

Como ya mencionó, uno de los principales objetivos al escribir programas paralelos es lograr un mayor rendimiento de la solución propuesta ¿Cómo podemos evaluar un SP?

Para la evaluación del desempeño o *performance* de un SP es necesaria la definición de métricas de rendimiento. Como el enfoque de la computación paralela es resolver grandes problemas en un tiempo relativamente menor, los factores que contribuyen al logro de este objetivo son, por ejemplo el tipo de hardware utilizado, el grado de paralelismo del problema y qué modelo de programación paralela se adopta. El rendimiento se logra analizando y cuantificando los datos, los procesos, las comunicaciones, etc. Para ello, existen varios parámetros de evaluación de rendimiento, los más destacados son: Aceleración, Eficiencia y Escalabilidad. La determinación de cada uno de ellos se realiza de la siguiente manera:

- *Aceleración:* o SpeedUp indica el beneficio o ganancia de resolver un problema en paralelo. Es la relación entre el tiempo para resolver el problema en secuencial (TS) y el tiempo requerido por el SP en *p* unidades de procesamiento (TP), Saccone (2015) . La aceleración es definida entonces como:

$$S = \frac{TS}{TP}$$

Se dice que existe una aceleración lineal cuando $S = p$. Esto significa que la velocidad de ejecución aumenta según el número de procesadores. Por supuesto, este es un caso ideal. Se pueden tener los siguientes casos:

- $S = p \Rightarrow$ La aceleración es lineal o aceleración ideal.
- $S < p \Rightarrow$ Estamos en presencia de una aceleración real. Generalmente es el caso habitual, ya que en todo SP existen tareas que si o si deben realizarse en secuencial e implican tiempo.

- *S > p* es aceleración superlineal. Son situaciones excepcionales, generalmente se da en los casos en que existen optimizaciones de los compiladores y heterogeneidad de la arquitectura.

Se denomina aceleración absoluta cuando TS es el tiempo de ejecución del mejor algoritmo secuencial, y es relativa cuando TS es el tiempo de ejecución del algoritmo paralelo para un sólo procesador.

- *Eficiencia:* Un resultado ideal para un SP ejecutando en *p* procesadores es obtener *S = p* (Almeida et al, 2008). Sin embargo, esto rara vez ocurre. Por lo general, se insume tiempo para permanecer en estado ocioso, comunicaciones (sincronizaciones) o cómputo secuencial. La eficiencia es una medida de rendimiento que calcula qué tan bien utilizados están los procesadores, en comparación con la cantidad de esfuerzo que se desperdicia en la comunicación y la sincronización.
  Se define a la eficiencia como:

$$E = \frac{S}{p} = \frac{TS}{p\ x\ TS}$$

  Los SP cuya aceleración es lineal logran el valor máximo de eficiencia, o sea *E = 1*; en otros casos el valor de *E < 1*. Los casos se identifican son:
  - E = 1, aceleración lineal.
  - E < 1, generalmente es el caso real.
  - E>1, aceleración superlineal.
  - E << 1, es un problema que es paralelizable con baja eficiencia.

- *Escalabilidad:* Es la capacidad de ser eficiente respecto al crecimiento del problema o de los recursos de hardware. Un SP es escalable cuando ante incrementos de alguno de los dos factores, se mantiene la eficiencia o se mejora (Almeida et al, 2008)(Saccone, 2015).
  En términos generales, una tecnología es escalable si puede manejar problemas de tamaño cada vez mayor. Un SP es escalable si para cualquier tamaño de entrada y número de procesadores se mantiene la eficiencia obtenida, es más, esta mejora cuando el tamaño del problema o los recursos computacionales crecen (Pacheco, 2011).

Las limitaciones de un cómputo paralelo son introducidas por la ley de Ahmdal. (Pacheco, 2011) (Saccone, 2015). La ley de Amdahl es una ley ampliamente utilizada para diseñar procesos y algoritmos paralelos. Establece que la aceleración máxima que se puede lograr está limitada por el componente en serie del programa:

$$S = \frac{1}{1 - P}$$

Donde *1 - P* denota el componente en serie (no paralelo) de un programa. Es decir, a menos que prácticamente todo un programa en serie esté paralelo, la posible aceleración será limitada, independientemente de la cantidad de procesadores disponibles. Supongamos que podemos paralelizar el 90% de un programa, aunque la paralelización sea *"perfecta"*, es decir, para dicha parte, independientemente del número de procesadores usados, la aceleración va a ser lineal; el 10% restante debe ser resuelto secuencialmente (por ejemplo: lectura de datos, impresión, etc.), entonces el máximo alcanzable de la aceleración es el 90% de la cantidad de procesadores (Pacheco, 2011)(Saccone, 2015) .

En este trabajo evaluamos las distintas métricas aquí expuestas para los casos de estudio en distintos escenarios.

2.2. Autómata Celulares y Simulación

En el campo de las matemáticas es vital poder crear y asociar herramientas con el objetivo de explicar los fenómenos que nos rodean. Generalmente, estas representaciones se logran mediante modelos matemáticos que consisten tradicionalmente en plantear, resolver ecuaciones representativas y darnos una posible respuesta a dichos fenómenos. En los avances adquiridos en el estudio de sistemas dinámicos podemos destacar a los autómatas celulares (AC) (Caligaris y Rodríguez, 2010)(Reyes Gómez, 2011).

Los AC surgen alrededor de la década de 1940 con la intervención de John Von Neumann, quien intentaba modelar una máquina capaz de autoreplicarse. Al final de su investigación logró obtener un modelo de dicha máquina con reglas aplicables a una red rectangular. Inicialmente fue interpretado como células, las cuales nacían, crecían, se reproducen y mueren (Reyes Gómez, 2011) (Caligaris y Rodríguez, 2010).

Según Reyes Gómez (2011), podemos definir a un AC como: *"Un modelo matemático para un sistema dinámico, compuesto por un conjunto de celdas o células que adquieren distintos estados o valores. Estos estados pueden cambiar de un instante de tiempo a otro. El tiempo es discreto, es decir, se puede cuantificar con valores enteros a intervalos regulares. De esta manera este conjunto de células logran una evolución según una expresión matemática sensible a los estados de las células vecinas. Este cambio se le conoce como regla de transición local"* (página 4).

A partir de esta definición es posible plantear vía la simulación, a un AC como un sistema dinámico discreto conformado por un conjunto finito de células con un estado o configuración inicial, las cuales evolucionan con el paso del tiempo sobre un espacio regular (según la naturaleza del fenómeno puede ser finito o infinito) de forma síncrona. Se puede describir a un AC como una séxtupla (L, V, Q, L_0, f, r) donde:

- *Espacio celular (L):* es una retícula o lattice *d*-dimensional, donde tienen lugar las evoluciones, cada división homogénea es una célula o celda, de ahí el nombre de autómata celular. Si *d* = 1, la retícula es un vector, si *d* = 2, es una matriz, etc. En teoría, el espacio puede extenderse infinitamente; sin embargo, para fines prácticos, se requiere tomar ciertas consideraciones y adaptar la definición original considerando retículas finitas en donde las celdas del AC interactúan. Esto implica prestar especial atención a aquellas celdas residentes en los bordes/frontera de la retícula (Caparrini, 2016). Existen distintos tipos de fronteras, las más comunes son:
    - *Frontera abierta*: todas las celdas fuera del espacio del AC tienen un valor fijo.
    - *Frontera reflectora*: Las celdas fuera de la retícula reflejan los valores de aquellas dentro de la misma, es decir una celda de la frontera toma como valor el de la celda del borde, dentro de la retícula.
    - *Frontera periódica:* las celdas de la frontera interaccionan con sus vecinos inmediatos y con las celdas en el extremo opuesto del espacio. En una retícula 1-dimensional, esto se visualiza como un anillo.

- *Sin fronteras:* la representación del autómata no tiene límites. Esto puede aproximarse haciendo uso de implementaciones dónde se hace crecer dinámicamente la memoria de la retícula; cada vez que las celdas interactúan con celdas fuera de la retícula, ésta se hace más grande para dar lugar a estas interacciones. La retícula comienza con un tamaño definido y finito, y conforme se requiera va creciendo en el tiempo. Con esta condición sólo se pueden inicializar las celdas dentro de la retícula inicial finita.

  No hay que confundir esta condición de frontera con la definición original de AC cuya retícula es inicialmente infinita. Obviamente existe un límite para esta condición, el cual es impuesto por la memoria disponible.

La selección del tipo de frontera depende de a modelar. Por ejemplo, si se desea modelar un sistema de evacuación de personas en casos de incendios de edificios, convendría usar la frontera abierta, donde los límites del AC representarán las paredes del edificio.

- *Vecindario (V):* Es el conjunto finito de celdas que definen la vecindad para una AC. Puede ser expresado como una función, donde a cada celda le corresponde un conjunto de sus vecinos. Es la región espacial necesaria para que una celda determine su estado futuro. En principio, no existe restricción sobre el tamaño de la vecindad, pero sí debe ser definido de forma análoga para todas las celdas. La cantidad de celdas y la forma del vecindario puede variar según la necesidad del sistema pudiendo cambiar a lo largo del ciclo de vida de la simulación, sin embargo, en la práctica generalmente está formado por la celda a analizar y un conjunto acotado de celdas alrededor de ella. Si la vecindad es demasiado grande, la complejidad de las reglas de evolución pueden ser excesivas (por lo general, la complejidad crece en forma exponencial en función al número de celdas vecinas).

  Para autómatas celulares 2-dimensional existen dos vecindarios muy utilizados en la literatura; ellos son:

  - El vecindario de Von Neumann: donde considera como celdas vecinas a las cuatro ubicadas de forma ortogonal, son aquellas que tocan un lado de la celda.

- El vecindario de Moore: toma en consideración no sólo las ortogonales, sino también todas las celdas ubicadas en las diagonales. Son ocho las celdas vecinas de cada celda.

Generalmente se consideran vecindarios cuadrados para cada celda del autómata, pero esto puede no ser así, pueden tomar otras formas: triángulos, hexágonos, etc., todo depende de las características del problema a resolver.

- *Estado de las celdas (Q): es el conjunto finito de todos los posibles estados de las celdas. En cada instante de tiempo, el AC presenta un número finito de estados y cada célula toma un valor de este conjunto de estados*

  Los autómatas más básicos son los binarios, donde cada autómata puede estar en uno de dos estados 0 o 1; blanco o negro; vivo o muerto; etc.. Bajo este concepto, el AC tal vez más conocido es "el juego de la vida" diseñado por el matemático británico John Horton Conway, en donde las celdas pueden estar en uno de dos estados, vivas (negro) o muertas (blanco). En la sección 2.2.2. Veremos con más detalle este AC.
- *Configuración inicial (L_0):* es la asignación inicial de estados para cada celda de L. Cada una de las células del espacio parte desde un estado de evolución inicial del sistema. Esta configuración puede influir en gran medida en la evolución del AC a lo largo del tiempo.
- *Reglas de Evolución o Función Local (f):* es la función de transición que asigna un nuevo estado a una celda teniendo en cuenta el estado de todos sus vecinos.

  Las reglas de transición (o evolución) definen la dinámica del sistema. Para cada celda en un instante determinado, calcula el nuevo estado dependiendo del estado inmediatamente anterior de su vecindad. Sabiendo que cada elemento de la retícula es un autómata, entonces estas reglas están definidas como la función de transición del mismo, especificadas en una tabla de reglas, con una entrada para todas las posibles configuraciones del vecindario. Si *K* es el número de estados posibles y *n* el número de celdas en el vecindario, entonces habrá *Kn* posibles reglas de transición del AC. Las reglas de transición pueden ser:

  - Deterministas: partiendo de la definición estricta, un AC es un sistema determinístico si la regla es una función bien definida y una particular condición inicial dada evoluciona siempre del mismo modo. En función de las celdas utilizadas para el cálculo tenemos (Martínez, 2006):

- *Reglas totalitarias:* se suman los valores de los elementos que forman la vecindad y todas aquellas vecindades correspondientes a esa suma, evolucionan al mismo valor.
- *Reglas semi-totalitarios:* se realiza una suma, pero sólo con los vecinos, no se tiene en cuenta la celda central (la cual es considerada solamente para determinar en qué condición debe evaluarse la suma).

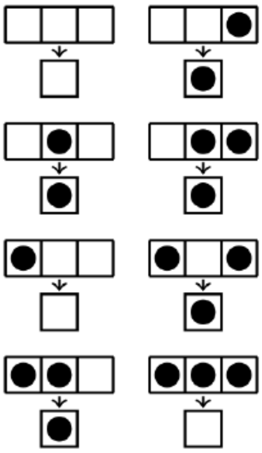
- *Probabilistas*: relajando un poco la condición de determinismo, existen sistemas donde es conveniente tener cierto grado de aleatoriedad en las reglas. El caso más común es que basado en la información de la vecindad, se obtiene una probabilidad *p*, la cual le permite a una celda cambiar a un estado u otro (o incluso no cambiar de estado). Esto da lugar a los llamados autómatas celulares estocásticos.

Si bien los AC deterministas son más sencillos y son suficientes para ciertos problemas, existen otros modelos que para lograr mayor exactitud es necesario considerar aspectos probabilísticas en su dinámica. De esta manera, se pueden conseguir sistemas que alcancen un comportamiento deseado pero con cierta aleatoriedad y realismo (Guedez, 2005).

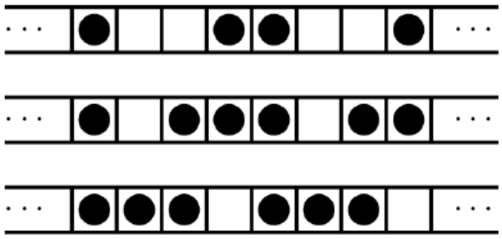
- *Reloj virtual (r):* Determina cuándo debe evolucionar el autómata según las reglas de evolución, haciendo que todas las celdas cambien de estado al mismo tiempo.

Los AC con el tiempo no sólo han sido usados para diferentes aplicaciones como por ejemplo para modelar problemas de genética (Biología); para estudiar problemas de dinámica de fluidos (Física); para el estudio cinético de las reacciones (Química)(Caligaris y Rodríguez, 2010), sino también han incorporado otras técnicas en su desarrollo como son el procesamiento paralelo y de imágenes (Reyes Gómez, 2011).

En la Figura 2.11. se muestra un ejemplo de AC 1-dimensional, en 2.11.a. se detallan las reglas de evolución y en 2.11.b. la evolución del AC desde el estado inicial en el tiempo *t* a los tiempos t+1 y t+2.



(a) Reglas de Transición.



(b) Evolución del autómata en 3 tiempos discretos

Figura 2.11. Ejemplo de Autómata Celular
Fuente: Suarez Alvarez, Mariano (2011). Autómatas Celulares. Departamento de Matemática Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. Buenos Aires, Argentina. Página 12

Como la simulación es un proceso que permite observar cómo evoluciona el sistema a lo largo del tiempo, permitiendo entender mejor el problema de estudio y predecir con mayor precisión eventos futuros, y los AC modelos simples para representar sistemas dinámicos y complejos; es posible deducir que los AC son adecuados para la simulación de estos sistemas. Permiten representar sistemas que evolucionan a través del tiempo de forma discreta; simulan la evolución temporal de diferentes modelos mediante unas pocas reglas simples, sin importar la complejidad del sistema, pudiendo observar las funciones básicas del mismo. Dentro de la Vida Artificial, son el más claro ejemplo de la búsqueda de la inteligencia; por ejemplo, si un AC representa un sistema de la evolución de la vida, podríamos observar el nacimiento, la alimentación y la muerte de las especies del sistema. La gran ventaja de éstos radica en que sólo es necesario identificar el comportamiento global del sistema y, a partir de allí, deducir un conjunto de reglas de evolución. Son considerados herramientas muy útiles en la construcción de modelos donde los elementos del sistema son de similar naturaleza y comportamiento.

2.2.1. Origen de los Autómatas Celulares

Sabiendo qué son los Autómatas Celulares y cómo es su funcionamiento, en esta sección vamos a mostrar sus orígenes El concepto fundamental fue propuesto alrededor de los años 40 por John Von Neumann como un intento de modelar el comportamiento del cerebro humano para resolver problemas complejos. La idea básica era poder crear robots capaces de construirse a sí mismos, conocido como modelo cinemático. Sin embargo, el costo que implicaba proveer al robot de una suficiente cantidad de partes a partir de las cuales pudiera armar a sus congéneres (autoreproducirse) fue un problema que se presentó de inmediato.

Por su parte, Stanislaw Ulam, había estado estudiando el fenómeno de crecimiento de cristales en una red lattice (una red infinita, desplegada como un tablero de ajedrez, donde cada cuadrado podía ser visto como una ‘celda’), le sugirió a Von Neumann que utilizase un método matemático abstracto para simular a sus robots. Así, Neumann replanteó su Autómata Autorreproductor, siendo este conocido como el primer Autómata Celular (AC). Dicho autómata fue definido como una matriz bidimensional donde cada celda contenía un robot, cada celda era una máquina separada de estados finitos, la cual actuaba de acuerdo a un conjunto compartido de reglas y a su vecindario.

Hay autores que dividen la historia de los Autómatas Celulares en tres: la primera etapa surge con el autómata de Von Neumann; la segunda se ve representada con la importancia adquirida en éstos a raíz del juego de la vida (Gardner, 1970); la tercera etapa llega a mediados de los años 80’s con Stephen Wolfram, quien se encaminó en el estudio de los autómata celulares en una dimensión.

Wolfram es un científico inglés nacido en 1959 que inició sus investigaciones en AC aplicando varios conceptos de dinámica no-lineal y mecánica estadística. Como ya en ese entonces se contaba con varias facilidades gráficas, se podían realizar muchos estudios experimentales. Esto permitió a Wolfram ir más allá: mientras que en los modelos propuestos por Von Neumann y Conway, las funciones representaban números muy grandes y cada autómata tenía un propósito específico para resolver un problema, en el estudio de Wolfram el método era sistemático, tomando un conjunto de reglas y estudiando sus evoluciones. Así observó que dada una regla, el AC exhibía diferentes comportamientos para diferentes configuraciones iniciales. Esta idea lo llevó a una búsqueda para establecer una clasificación de acuerdo a las evoluciones de los autómatas, conocida como las “clases de Wolfram”. Wolfram caracterizó a los AC de acuerdo a sus comportamientos y a las configuraciones que adoptan, es decir en función de la configuración inicial y las reglas. Así, tenemos algunos AC que se estabilizan, permaneciendo en una configuración; otros exhiben comportamientos periódicos, mientras que otros tienen comportamientos caóticos y aparentemente imprevisibles.

2.2.2. El Juego de la Vida

John Horton Conway nació en 1937, comenzó su carrera de Matemáticas en la Universidad de Cambrige. A finales de 1960 Conway, retomó el trabajo de Von Neumann, su idea de construir una máquina que fuera capaz de crear una copia de sí misma haciendo uso del concepto de vecindad de un autómata definido por Moore. Diseñó y dio a conocer al autómata celular más famoso: The Game of Life. Fue así llamado por su relación con la modelación de sistemas biológicos, tal como: ecosistemas, incendios forestales, comportamientos colectivos de seres vivos, entre otros. Su primera publicación fue en un artículo de Martín Gardner en “Scientific American” en la sección de juegos matemáticos (Gardner, 1970).

Este AC tiene lugar en un tablero de dos dimensiones (arreglo bidimensional) aunque se puede construir en una, dos o más dimensiones(Adamatzky, 2010). Cada celda que lo conforma posee hasta 8 vecinos directos, definido según el vecindario de Moore.

Una celda puede estar en uno de dos estados *Viva* o *No Viva.* Las reglas de transición de un estado a otro se definen como: En las unidades de tiempo discretas llamadas generaciones o tiempo evolucionan; aplicamos una serie de reglas a la configuración actual para llegar a la configuración de la próxima generación; para este AC en particular, se usan las siguientes reglas:

- Si una célula *Viva* tiene 2 o 3 células vecinas vivas, entonces permanece *Viva* la próxima generación, de lo contrario, muere(*No Viva*) por soledad: 1 sola celda vecina viva, o por sobrepoblación: más de 3 vecinos vivos.
- Si una célula No Viva tiene exactamente 3 células vivas, cobra vida la próxima generación.

Muchas veces, los reglamentos establecidos definen el tipo de AC con el que estamos tratando. El juego de la vida en particular tienen innumerables aplicaciones en realidades de juegos de mesa como el famoso juego, de origen chino “GO” y en la representación reproductiva de bacterias (Białynicki-Birula y Białynicki-Birula, 2004).

Dentro de la escala evolutiva de los AC, a través del correr de generaciones, existen patrones básicos que no son más que configuraciones de vecindades de células que determinan un comportamiento concreto con un número mínimo de células vivas en la estructura. Algunos de estos patrones se generan con facilidad de forma instantánea durante el desarrollo de juego y otros son interesantes por su comportamiento.

Los patrones básicos según nivel de complejidad son:

- *Estáticos o inmortales:* Se trata de patrones continuos que debido a su configuración estable permanecen inalterados al correr del tiempo. La Figura 2.12. se ilustra una de las configuraciones en las cuales, las celdas permanecen en un estado estático por generaciones.
- *Oscilatorios*: Es un patrón que evoluciona cambiando de forma, pasado un número finito de generaciones, regresa a su forma original como se muestra en la Figura 2.13.

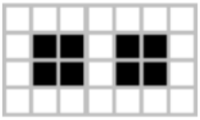


Figura 2.12. Patrón Estáticos o Inmortales.
Extraída de: López Salinas, Ana Miriam (2011). Introducción a la Vida Artificial y Autómatas Celulares. Facultad de Estudios Superiores Acatlán, UNAM. Página 10.

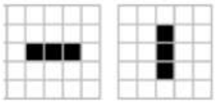


Figura 2.13. Patrón Oscilatorios.
Extraída de: López Salinas, Ana Miriam (2011). Introducción a la Vida Artificial y Autómatas Celulares. Facultad de Estudios Superiores Acatlán, UNAM. Página 11.

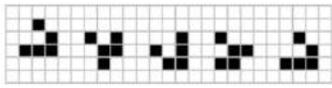


Figura 2.14. Patrón Astronaves.
Extraída de: López Salinas, Ana Miriam (2011). Introducción a la Vida Artificial y Autómatas Celulares. Facultad de Estudios Superiores Acatlán, UNAM. Página 12.

- *Astronaves*: O spaceships (Figura 2.14) son patrones que se caracterizan por desplazarse a través del tablero a lo largo del tiempo, bien sea de forma diagonal, horizontal o vertical.
- *Patrones reproductores:* Son patrones que aumentan constantemente su tamaño de población (células vivas).
- *Patrones inestables:* Son patrones que evolucionan a través de una secuencia de estados y nunca vuelven a su estado original. Existen también patrones que tardan mucho tiempo en estabilizarse y son llamados "Methuselahs”.

Todas y cada una de estas características hacen que el juego de la vida de Conway sea computacionalmente completo (este AC presenta algunas de las características buscadas por Von Neumann de una manera más simple, pues sólo utiliza dos estados).

Lopez Salinas (2011), destaca el artículo elaborado por Martin Gardner (en 1970); en base al trabajo de Conway, dicho artículo inspiró a comunidades científicas en todo el mundo a experimentar con el Juego de la Vida.

En este trabajo consideramos a los AC para llevar a cabo la simulación del sistema a estudiar, es por eso que se detalla el Juego de la Vida como el punto de partida a tener en cuenta para nuestros desarrollos.

2.3. Modelos de Propagación

En el relevamiento de modelos de propagación, la mayor parte de las investigaciones coinciden en considerar como al modelo de difusión/ propagación/ dispersión SIR. Fue en 1927 cuando Kermack y McKendrick propusieron un modelo que se convirtió en la base del modelado de epidemias moderno, conocido como modelo epidemiológico SIR. El mismo considera un escenario hipotético donde una enfermedad se desarrolla a lo largo del tiempo y en donde están involucrados únicamente tres clases de individuos: los Susceptible (Susceptible), Infectado(Infected) y Recuperado (Recovered) (Kermack, McKendrick y Walker, (1997). La Figura 2.15. muestra la transición entre los distintos estados del modelo SIR.

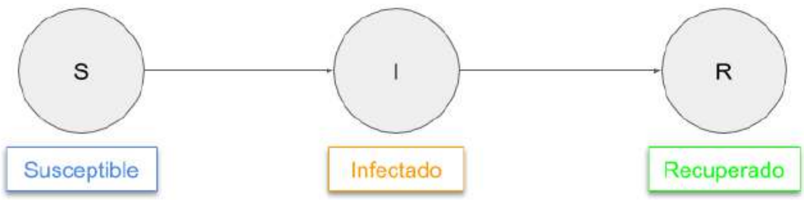


Figura 2.15. Modelo de Epidemiología SIR.
Fuente: Brauer, Fred y otros (2015). Modelos De La Propagación De Enfermedades Infecciosas. Universidad Autónoma de Occidente. Cali, Colombia. Página 12.

El enfoque del modelo SIR ha impactado positivamente en el área de modelado y control de epidemias, extendiéndose a otras problemáticas. Esto obedece, entre otras características, a su simplicidad, su valor didáctico, su aplicabilidad a datos reales y a su extensibilidad para el estudio de epidemias con mecanismos más complejos (Brauer et al, 2015).

El modelo se aplica sobre una población en la cual hay un número determinado de miembros que padecen alguna enfermedad infecciosa en estudio. Dicha enfermedad puede ser transmitida, en algunos casos, a otros miembros en contacto, de la misma población. El objetivo que se persigue es determinar ¿Qué porcentaje de la población total estará infectada y por cuánto tiempo?

Inicialmente la formulación del modelo empieza clasificando en 3 subclases identificadas con las letras S, I y R (Ver Figura 2.15.). Cada una representa el estado o clase en que está el individuo en el instante *t.* Así:

- S(t) denota el número de individuos susceptibles de contraer la enfermedad en el tiempo *t*.
- I(t) el número de individuos capaces de transmitirla; es decir, individuos infectados.
- R(t) es el número de individuos que han perdido la posibilidad de ser infectados; ya sea por haber sido:
    - aislados del resto,
    - inmunizados, se curaron y no vuelven a enfermarse más, o
    - fallecidos a consecuencia de la enfermedad.

  R(t) es entonces el número de individuos recuperados en tiempo *t*. Aunque los factores por los cuales un individuo entra a este último estado pueden ser diversos desde un punto de vista epidemiológico.

El flujo de transiciones de un grupo a otro se puede observar en la Figura 2.15. Además, para una población constante *N*, aunque los individuos van pasando de un estado a otro; a lo largo de todo el sistema N = S(t) + I(t) + R(t).

Brauer et al. (2014) mencionan en su libro al clásico modelo de Kermack y McKendrick, que ha influido en las últimas décadas. A partir del modelo SIR, se formularon variantes, las cuales se han convertido en prototipos de sistemas no lineales, utilizado para explicar, en un principio, las epidemias en salud (enfermedades transmisibles) en fenómenos biológicos. Entre los modelos derivados encontramos:

- *Modelo SIS:* Un individuo, al recuperarse, queda automáticamente susceptible para enfermarse nuevamente.
- *Modelo SIRS:* Existe un período de recuperación donde la persona es inmune a la enfermedad, pero luego de un tiempo el sistema inmunológico se debilita y vuelve a ser susceptible. Es ilustrado en la Figura 2.16.

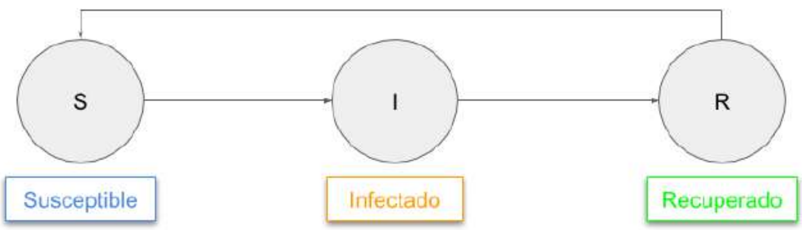


Figura 2.16. Modelo de Epidemico SIR.

Fuente: Brauer, Fred y otros (2015). Modelos De La Propagación De Enfermedades Infecciosas. Universidad Autónoma de Occidente. Cali, Colombia. Página 12.

- *Modelo SEIRS: A*ntes que un individuo pase a estar enfermo, pasa por un estado de incubación E (exposed), donde todavía no presentan síntomas.

Existen muchas modificaciones más del modelo, e incluso agregados como la posibilidad de que existan nacimientos y muertes dentro de la población estudiada. El estudio de estas exceden el alcance de este trabajo.

El uso de modelos dinámicos aporta una nueva dimensión permitiendo a teóricos y prácticos la posibilidad de formular nuevas preguntas dentro de un marco que permite explorar el impacto de intervenciones en la dinámica de los fenómenos de transmisión, por ejemplo las enfermedades contagiosas. Este proceso ayuda a identificar, cuantificar, evaluar e implementar políticas de intervención dirigidas a reducir el efecto de la epidemia o brotes.

2.3.1. Noticias como Contenido e Influencia para Receptores

La difusión de noticias puede verse como un fenómeno de propagación y, en consecuencia, aplicar lo visto en la sección anterior para su modelado. Con más detalle abordaremos ese tema en el capítulo siguiente. Ahora, veremos las características básicas de las noticias y sus medios de difusión actuales.

Las diferentes tecnologías cumplen un papel destacado en la vida cotidiana de las personas, las atrae a la desconexión de la vida real y provee de diferentes espacios *online* donde pueden compartir información. De esta manera logran una interacción y conexión social significativa mediante el teléfono móvil, el correo electrónico y las redes sociales entre las tecnologías más comunes de acceso a internet.

Esta posibilidad de acceso masivo a Internet, produjo un cambio de paradigma.

En el artículo de López-Borrull, Alexandre; Vives-Gràcia, Josep y Joan-Isidre, Badell (2018) se expresa que “Tradicionalmente los profesionales de la información han tenido a su alcance una serie de fuentes de información que se consideraban fiables y a las que podían recurrir para dar respuesta a las necesidades de sus usuarios. Una fuente confiable, de referencia, seguía una serie de controles y validaciones que permitían garantizar la calidad de la información que se podía encontrar. A partir de la irrupción de internet, del blog, la web, etc., el monopolio de comunicar información se rompe y ya no sólo los editores de periódicos, revistas, libros o medios de comunicación tendrán la capacidad de emitir sus mensajes" (p 2).

En el mundo online, donde las noticias se difunden de manera generalizada a alta velocidad y aparecen términos como diarios digitales, portales de farándula, podcasts, influencer (personas que usan la redes sociales para compartir material audiovisual con sus seguidores), no todo el contenido cumple con la propiedad de veracidad (Gómez Ramírez, 2015). Establecer cuán rápido se difunde una noticia, cualquiera sea su veracidad, puede resultar importante en determinadas ocasiones y contextos.

Si bien las noticias falsas (Fake news), las historias fabricadas, trolls, rumores o clickbait[2] no son novedad; en este tiempo se han diferenciado contextualmente por el potencial de su circulación por redes sociales (Delmazo y Valente, 2018). A continuación describimos las siguientes características de dos de estos conceptos, ellas son :

- *Fake news* son un producto pseudoperiodístico lanzado por portales de noticias, prensa escrita, radio, televisión, en redes sociales o, simplemente, en el boca en boca, con el objetivo de desinformar y engañar para influenciar manipulando las opiniones personales. Por lo general se la relaciona con material político y con titulares de muy fuerte impacto para causar curiosidad, haciendo que los consumidores crean esa información errónea.

  Un concepto clave en relación a la noticias falsas es la "pos-verdad" o también conocida como mentira emotiva (o moderna), la cual se emplea para distorsionar la verdad e inducir comportamientos como el voto en una elección, siguiendo con el ejemplo de distribución de material político, en base a hechos falsos.
- *Los Trolls* son aquellos entes que resguardan su identidad y generan las noticias falsas con el propósito de generar molestias (Delmazo y Valente, 2018) (Espinoza et al, 2019).

[2] Titulares que son cebos de clicks. (Delmazo y Valente, 2018)

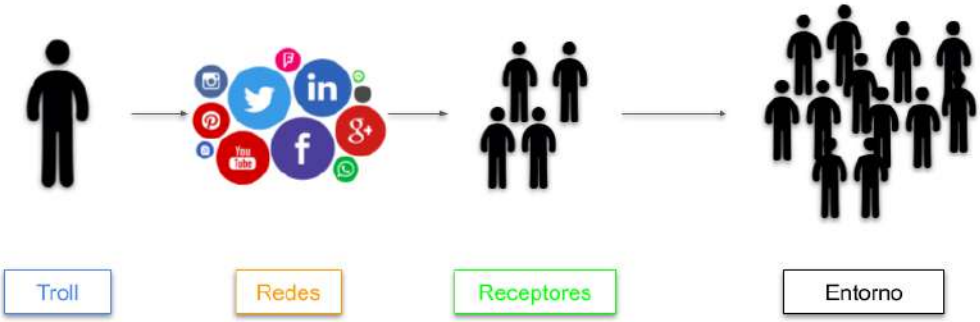


Figura 2.17. Siembra del contenido digital falso y dispersión.

Las redes sociales persiguen el objetivo de ser el lugar central de encuentro de las personas, permitiéndoles realizar publicaciones de: negocios, pensamientos, material de texto o multimedia de algún evento de su vida y otros. Muchas veces las incorporamos a nuestras relaciones sociales presenciales o interpersonales, como un elemento más de nuestra vida. Es por ello que la información ligada al periodismo o comunicación mediática distribuida a través de redes sociales u otro medio masivo puede carecer de veracidad en el sentido de datos que aportan o exhiben, dónde no se pueden verificar o corroborar y quién la generó. La Figura 2.17. muestra este fenómeno de difusión.

En la difusión de una noticia influyen varios factores, entre ellos se destacan el nivel de confianza interpersonal con los individuos del entorno, el grado de influencia de la persona, su edad, condición social, convicciones, formación, etc. En este trabajo nos enfocamos en el grado de confianza y la intención de propagación.

2.4. Modelo SIR con Autómatas Celulares

Si bien el modelo SIR permite predecir de forma efectiva cómo se comporta una población frente a una infección mediante el uso de ecuaciones diferenciales (determinando ciertos parámetros), lo cierto es que si se quiere representar de forma analítica una realidad más compleja, se deberían modificar dichas ecuaciones y agregar variables a fin de representar todas las variantes de la realidad. Esto puede llegar a ser muy complejo, o incluso imposible, ya que asumen una población homogénea, impidiendo tratar el proceso infeccioso en toda su magnitud, sin poder representar correctamente características de personas, medio ambiente o interacción de forma dinámica.

Por tal motivo, en muchas ocasiones es recomendable utilizar simulaciones. Las mismas presentan además ventajas propias de la técnica: a diferencia de las ecuaciones diferenciales que funciona como una caja negra, donde dado ciertos valores de entrada, se obtienen determinados valores de salida, la simulación permite observar la evolución del sistema en el tiempo, entender su comportamiento y, quizás, poder predecir eventos futuros.

Por las características enunciadas de los AC, el paso del modelo SIR al modelo de AC es prácticamente directo, simplemente se deben encontrar las reglas adecuadas para representar la realidad. En función de lo dicho, se puede afirmar que resolver el modelo SIR mediante simulaciones que permitan observar la evolución de la enfermedad o del fenómeno de difusión en un AC y cómo se dispersa en una población determinada es mejor que hacerlo utilizando ecuaciones diferenciales, debido a que permite trabajar con poblaciones de índole heterogénea y otorga la posibilidad de describir en detalle la totalidad del escenario (medio ambiente y características de los individuos).

En este trabajo vamos a trabajar con el modelo SIR y AC para modelar el sistema de propagación de noticias. Dadas las características de los AC (La evolución de cada celda sólo depende de su vecindario actual), es posible aplicar técnicas HPC en la solución. Todo esto se presenta en el siguiente capítulo.

2.5. Aplicaciones

Los autómatas celulares son ideales para modelar y simular cualquier sistema dinámico, el cual evoluciona en el tiempo; el único requisito a tener en cuenta es la discretización de dicho sistema y cómo abstraer la realidad para convertirla en un modelo computacional.

En función de esto, los AC se han utilizado para innumerables aplicaciones y estudios. Si bien muchos no se pueden catalogar, los ejemplos los dividimos por la rama de la ciencia en la que se han aplicado (Reyes Gomez, 2011) (Peredo y Ramallo, 2003). Entre ellos podemos encontrar:

- Física y Química

  Los AC son estructuras ideales para la representación de algunos sistemas físicos y químicos complejos; ellos permiten modelar su comportamiento mediante reglas y ver su evolución, sin la necesidad de recurrir a ecuaciones matemáticas complejas.

  En física, podemos tomar como ejemplo las interacciones entre partículas, formación de galaxias, dinámica de fluidos, o incluso sistemas matemáticos tan complejos como la teoría del caos. En química se han utilizado para estudiar sistemas de difusión, los cuales describen cómo una o más sustancias distribuidas en el espacio cambian bajo la influencia de dos procesos: reacciones químicas locales en las que las sustancias se transforman unas en otras; y difusión, donde las sustancias se expanden en el espacio.

- Biología y Bioinformática

  Desde la aparición del juego de la vida, la utilización de los AC orientados a los procesos biológicos ha ido en constante aumento. En la naturaleza existen infinidad de sistemas biológicos complejos, en los que se refleja la auto-organización, a partir de interacciones simples de las partes conduciendo a cambios en el tiempo y en el espacio (como por ejemplo: colonias de insectos, o el crecimiento de plantas). Esto los hace inadecuados para representarlos con ecuaciones diferenciales ordinarias, pero muy adecuados para representarlos mediante AC.

  A partir de las partes entrelazadas que generan propiedades nuevas y emergentes (y muchas veces oculta al observador y sin la posibilidad de explicarse analizando los elementos aislados), se deben identificar estados locales y reglas generales de evolución. Esto permite realizar una representación del modelo sólo conociendo el comportamiento global del sistema. En este caso, tenemos por ejemplo propagación de virus, epidemias y bacterias, comportamiento de glóbulos en el cuerpo, genética, etc..

- Arquitectura

  En esta área, se utilizan los AC para generar patrones o modelos a fin de sugerir formas arquitectónicas. Esta generación surge de la evolución natural del autómata en función que, a partir de un estado inicial, el autómata genera un nuevo estado (resultado) el cual sirve como base para el siguiente grupo de resultados. Este mecanismo (método de sustitución recursivo) continúa hasta llegar a una configuración final, ofreciendo así una interesante y rica plataforma de desarrollo de modelos arquitectónicos.

  En esta rama, los AC que más interesan son los tridimensionales. El análisis y proceso de cómo utilizar los AC en arquitectura es complejo y excede el alcance de este trabajo. Lo más importante del proceso no es el resultado en sí, sino cómo usar los datos generados en el patrón para interpretar y modificar los resultados para su uso en la arquitectura.

- Informática

  En esta área, los AC se utilizan para innumerables tareas, las cuales van desde el procesamiento de imágenes hasta la criptografía. Ésta última es realmente importante en la actualidad, ya que con todas las formas de comunicación electrónica existentes, es indispensable que la información se transmita de manera segura y confiable. Así, se pueden utilizar AC reversibles a nivel de bits, quienes a partir de una configuración inicial (el texto), devuelve el mensaje cifrado. Esta propiedad de reversibilidad de los AC’s consiste en que cada configuración tiene un ancestro u origen único, permitiendo regresar al origen, así haya transcurrido un gran lapso de tiempo. Así no hace falta combinar el mensaje cada vez que se quiera encriptar o desencriptar, sino simplemente aplicar la regla de evolución y después su inverso.

- Fenómenos Naturales: Incendios

  Los AC han sido muy utilizados para la simulación en casos de incendios. Por un lado, tenemos la ecología del fuego, rama de la ecología centrada en los orígenes de fuegos silvestres y su relación con el ambiente que lo rodea; por el otro, tenemos la relación existente entre el origen de fuego dentro de estructuras edilicias cerradas, con solo algunos puntos de evacuación posibles, y las relaciones existentes entre las dinámicas pedestres de evacuación de las personas dentro del edificio.

Este tipo de simulaciones es de verdadera importancia. En la actualidad pueden implicar la pérdida de vidas humanas, ya sea en el proceso de extinción de incendios forestales, como en la mala evacuación en caso de ser necesaria. Además, es prácticamente imposible realizar experimentos reales con fuego, por el peligro que esto conlleva. Claro que se pueden realizar simulacros de evacuación o experimentos con fuego controlado, pero la presencia de un peligro real (cuando verdaderamente existe un incendio) modifica el comportamiento de las personas, por lo que los experimentos tampoco representan la realidad de forma precisa.

La importancia de los AC en este tipo de simulaciones radica en que se pueden encontrar reglas básicas de propagación del fuego, en base a las condiciones y formas del terreno, logrando así simulaciones bastantes realistas simplemente brindando una configuración inicial, y analizando cómo mediante estas reglas el ambiente evoluciona.

Por todo lo expuesto, es posible establecer que los AC son una excelente herramienta de análisis, proveen facilidad y versatilidad no conseguida por otros métodos de simulación como son las ecuaciones diferenciales. Brindan una comprensión más detallada de la dinámica de los sistemas a investigar, son una herramienta de invaluable potencia a la hora de generar escenarios artificiales, que de otra manera serían excesivamente costosos o incluso imposibles de reproducir y de estudiar.

La cantidad de aplicaciones en las que se pueden utilizar van mucho más allá de las aquí mencionadas. La capacidad innata de los AC para simular fenómenos naturales, ayuda a una interpretación más completa de los resultados, nos permite lograr un mayor entendimiento de fenómenos del universo que nos rodea.

CAPÍTULO 3

Casos de Estudio y Resultados Experimentales

En la primera parte del capítulo se presentan los antecedentes bibliográficos consultados en referencia a la temática de AC, HPC y modelos propuestos para la simulación de divulgación de noticias/rumores.

El Juego de la Vida, el más popular AC conocido en la literatura, constituye la base para aspectos de diseño y desarrollo del modelo HPC-AC_{N-R} y su prototipo paralelo. Los aspectos básicos de diseño e implementación son detallados aquí, como así también diversos escenarios de pruebas recreando distintas realidades sociales. Los distintos escenarios fueron evaluados en un Cluster de computadoras según parámetros de desempeño estándares: aceleración y eficiencia alcanzada por prototipo, y parámetros sociales: noticias más difundidas, influencia de la densidad población en la difusión, de la distribución etc.

3.1. Antecedentes

Actualmente, bajo los ejes de investigación perseguidos en este proyecto, se puede destacar material bibliográfico concerniente a AC, los mismo se han utilizado como modelos para el campo de la simulación a partir de su creación en 1940. Desde “Game of Life” propuesto por John Conway (Gardner, 1970), utilizando una matriz y un AC elemental con celdas conteniendo uno de dos valores: 1 o 0, hasta otros usos más complejos como son por ejemplo: la configuración inicial de un sistema dinámico para criptosistemas, la modelización y simulación de las funciones cerebrales en el ámbito de la bio-informática, la generación patrones o modelos para sugerir formas arquitectónicas en la disciplina de arquitectura, entre otros (Reyes Gómez, 2011). También se los ha utilizado, con la aplicación de técnicas HPC en dinámica de fluido computacional, en este caso utilizando las Unidades de Procesamiento Gráfico (GPU) y en el estudio de la propagación de la Influenza estacional, para computadoras con memoria compartida (Kirk y Hwu, 2010).

Otro trabajo relacionado a un modelo de dispersión, aunque no especifica cómo fue realizado, es el expuesto en la conferencia TED (3 Abril del 2015) en la ciudad de Vancouver (Canadá) por el cofundador de Microsoft, Bill Gates, donde mostró una simulación, la cual reproduce el modelo de un virus que se propaga por el aire como la gripe Española en 1918, guardando relación con la actual pandemia de COVID-19, para más detalle ver APÉNDICE B.

En relación al trabajo aquí presentado, se encontraron propuestas para el estudio de divulgación de noticias falsas, donde se asimila la dispersión de las misma basados en modelos epidémicos y en la susceptibilidad de las personas a creer, o el grado de influencia que tienen de su alrededor (Xiaoxia, Jiajia; Yuhe, Mengmeng, 2018).

En (Brito do Nascimento, Takeshi Seo, 2017) encontramos una propuesta del modelo y desarrollo, definido empíricamente, basado en la información de grandes portales de noticias y observaciones diarias.

Otros aportes son los realizados por Yiran Gu y Jinzhu Ding (2012), en su investigación donde estudian el fenómeno de la propagación de rumores basado en la teoría de modelado de CA, definen dos parámetros con un rol fundamental: el grado de confianza entre las células, representando la razón de lazo entre ellas, y la intención de propagación de la célula. Establecidos estos parámetros, pudieron observar y analizar la influencia de los efectos de propagación ajustando los coeficientes. Sus resultados muestran que la distribución de los parámetros tienen cierta influencia en la propagación de los rumores y el grado de difusión disminuye con el aumento de la variación del grado de confianza y la intención de propagación. En dicho trabajo cada una de las células puede estar en uno de tres estados en el tiempos discreto *t*, ellos son: estado *H* (estado neutral, no conoce el rumor), estado *SS* (conoce el rumor y lo dispersa) y estado *SI* (sabe el rumor pero pierde el interés en él y no lo propaga).

Maitanmi, Olusola Stephen; Adekunle, Yinka y Agbaje, Michael (2013), también publican su contribución al estudio de la propagación de chismes o rumores sobre AC, utilizando un modelo estocástico. La propagación del rumor se ve alentada por el uso de células homogéneas que pueden permitir o rechazar el mensaje, es un modelo de paridad, las células blancas nunca escucharon el rumor mientras que, las células negras sí. Cada vecino se entera de una noticia por uno o más vecinos conocedores de ella. Una vez enterado puede difundir la novedad a sus vecinos. En cualquier instante de tiempo, existe la posibilidad de que, una célula blanca reciba la noticia/chisme desde una célula negra vecina y se vuelva negra. Una vez que una célula ha escuchado al menos una noticia/chisme, nunca se olvida, por lo que en el modelo, una célula negra nunca vuelve a ser blanca.

En este trabajo nos basamos en el trabajo realizado por (Gu et al, 2012) para aplicar técnicas HPC y ver el comportamiento del desempeño de la simulación en diferentes escenarios.

3.2 Casos de Estudio

En este capítulo se presentan dos casos de estudio con los cuales se trabajó, el Juego de la Vida (explicado en el capítulo anterior), y la propagación de noticias. Ambos casos fueron adaptados a un modelo general de AC que se diseñó, e implementó tanto en secuencial para CPU, como con técnicas de computación de alto desempeño, versiones paralelas, para computadoras paralela de memoria distribuida. Para cada caso se introduce brevemente sus principales características, funcionamiento general y detalles del AC asociado. Luego, se describe de manera general el modelo propuesto para AC en paralelo, y finalmente sus correspondientes implementaciones.

3.2.1. Juego de la Vida

A partir de las especificaciones formuladas en el capítulo anterior sobre el Juego de la vida, éste es directamente un AC, y a fines de especificar sus características según lo visto en el Capítulo 2, se puede definir como una 6-tupla $AC_{GOL}=<L,V,Q,L_0,f,r>$ de la siguiente manera :

- **Espacio celular *(L)*:** Es un arreglo bidimensional finito, con *frontera abierta*. Cada celda del autómata representa un lugar que puede ser poblado por una *célula*.
- **Vecindario (*V*):** Utilizamos el vecindario de Moore (FIGURA ---), que incluye a las 8 celdas que rodean a una celda central.
- **Estado de las celdas (*Q*):** Una celda puede estar en un estado de ***Q* = {*A*, *E*}** , donde:
  - ***A:*** Celda ocupada por una *célula* viva.
  - ***E:*** Celda vacía. En el caso de que una *célula* muera, deja libre la celda que ocupaba.
- **Configuración inicial (*L_0*):** Antes que la simulación comience, se debe especificar el tamaño de la matriz con la que se quiere trabajar, así como la cantidad de *células* vivas en la *población inicial*. En nuestro caso, la disposición de las *células* y la densidad poblacional se consideró de manera aleatoria.
- **Reglas de evolución (*f*):** En cada paso del tiempo, cada célula calcula la cantidad de vecinos vivos que tiene en su vecindario, actualizando luego su propio estado

en función de las *reglas de evolución*. A continuación, se describen las reglas originales de juego, conocidas como *Leyes genéticas de Conway*:

- **Supervivencia *(A→A)*:** Cada *célula* que cumpla el requisito de tener 2 o 3 vecinos vivos sobrevive a la siguiente generación (su estado se mantiene inalterado en el siguiente turno).
- **Fallecimiento *(A→E)*:** Una *célula* viva que tenga menos de 2 vecinos (*células* vivas en su vecindario) fallece por *aislamiento* o soledad en el siguiente turno. Una *célula* viva que tenga más de 3 vecinos vivos muere por *superpoblación* en el siguiente turno.
- **Nacimiento *(E→A)*:** Si una celda vacía pasa a tener en su vecindad exactamente 3 *células* vivas, su estado en el siguiente turno será el de *célula* viva (nacimiento de una nueva *célula*).

Estas reglas se simbolizan como “2,3 / 3” (condiciones de supervivencia / condiciones de nacimiento). Fueron las reglas originales formuladas por John Horton Conway, sin embargo, no son las únicas, ya que existen numerosas variantes.

- **Reloj virtual:** El tiempo es discreto. En cada intervalo de tiempo el AC evoluciona hacia una nueva generación de células, realizando una actualización exhaustiva de sus celdas.

De la formulación anterior, es claro que el Juego de la Vida es un juego de cero jugadores, basado en la evolución de estados sucesivos. Los estados futuros de las células dependen solamente de las condiciones del estado anterior del AC, no requiriendo la entrada de datos durante el desarrollo del mismo. El estado inicial y las reglas del juego son las que determinan el desarrollo del mismo. Como jugador, la participación en el juego únicamente consiste en determinar las condiciones iniciales de unas determinadas células, la población inicial o generación cero.

3.2.2. Modelo de Propagación de Noticias

Antes de formular el AC para el problema de la difusión de noticias-rumores o noticias-chismes, analizaremos el problema en sí.

Si bien la noticia y rumor son dos conceptos distintos, es posible encontrar encontrar alguna semejanza entre ambos. Para la (RAE, 2020), el significado de cada uno es:

- Noticia: Información sobre algo que se considera interesante divulgar.
- Rumor: Voz que corre entre el público.

Ambos se refieren a la divulgación de algún contenido "informativo" a un determinado público, por lo cual es posible establecer una gran similitud entre ellos. Basados en los estudios realizados y la bibliografía existente, es posible establecer un un modelo, el cual puede ser utilizado tanto para la divulgación de noticias como de rumores basados en AC.

En la bibliografía se hallaron variados modelos de dispersión de noticias, rumores o chismes, la mayor parte ellos resuelven el problema considerando estados binarios: el individuo(nodo) está en conocimiento del contenido en circulación y lo transmite, o desconoce totalmente su existencia. Otros modelos trabajan con un rango de estados similar al del modelo epidémico SIR: desconoce la noticia, la conoce y el tercer estado es cuando el individuo se encuentra desinteresado en creer y transmitirla. En nuestro trabajo nos basamos en este último enfoque, es decir, cada individuo de la realidad puede "*Conocer y transmitir*", "*Desconocer*" o "*Conocer, no estar interesado y no transmitir*" la noticia/rumor a otra persona.

De lo antes expuesto surgen varias preguntas, podemos distinguir: ¿Cómo decide si le interesa o no la noticia/rumor? En caso de querer transmitir ¿A quién le transmite lo que sabe?¿Cuántas veces comunica la novedad?. Responder todas estas preguntas nos definirá el trabajo del sistema.

La noticia le puede llegar a una persona de alguno de sus vecinos inmediatos. Como se mencionó en la Sección 2.2 del capítulo anterior, la vecindad es la interconexión de celdas más cercana a la celda(x, y). Existen muchos métodos para definir la vecindad, entre las más conocidas se encuentran la vecindad propuesta por Von Neumann (para una celda central, sus vecinos son las cuatro celdas ordenadas en forma de cruz como se puede apreciar en la Figura 3.1(a). Otra vecindad es la vecindad propuesta por Moore (Figura 3.1(b)), aquí el número de células vecinas a la celda central son todas aquellas que la rodean, siendo un total de 8 vecinos. Este vecindario puede constituir una muestra más real y precisa de lo que sucede a su alrededor.

(a) Vecindad de Von Neumann
Extraída de: [Fuente Figura 3.1(a)]

(b) Vecindad de Moore
Extraída de: [Fuente Figura 3.1(b)]

Figura 3.1 Vecindad de una Celda

Para nuestro simulador vamos a considerar la vecindad de Moore, esto significa que la celda central va a recibir noticias de cualquiera de sus 8 vecinos, pudiendo transmitir a cualquiera de ellos lo que sabe, si lo desea.

Una vez que una persona recibe una noticia/rumor, debe decidir si es de su interés o no, no sólo para conocerlo sino también para transmitirlo. La decisión la toma en función del *grado de confianza* o afinidad que tiene con la persona de la cual recibió la noticia y su intención de propagación. Si la evaluación de ambos supera un cierto límite, transmite la noticia a uno de sus vecinos, sino no hace nada con ella y la descarta. El grado de confianza y de intención son dos parámetros de cada individuo, para el primero existen tantos grados de confianza como vecinos tiene. Para el segundo, en cambio, es una característica individual de cada persona.

En base a la realidad enunciada, cada celda en el AC representa a una persona y la información que la define es mostrada en la Figura 3.2, donde se considera el estado(*estado*), el grado de confianza con sus vecinos (*v_interrelacional*), su intención de propagación (*grado_propagación)* y la versión de la noticia que manipula(*versión*), si es verdadera o falsa.

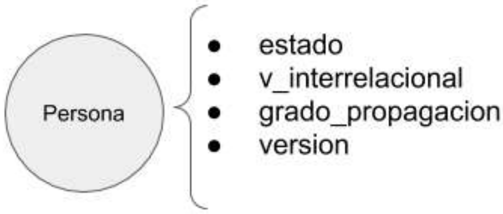


Figura 3.2. Modelo de Divulgación de Noticias

donde definimos al grado de confianza y la intención de propagación como:

- Grado de confianza: es el grado de fidelidad y dependencia entre los individuos (células). Se la define como:

$$b(ij) = x, x \in [0,1]$$

para $i \in [1, 8]$, $j \in [1, N]$ donde *N* es la dimensión del grid del AC.

b(ij) representa la confianza e intimidad de la celda *i* en cuanto a la celda *j*. Cuanto mayor es el valor de *b(ij)*, significa que *i* tiene mayor grado de confianza con *j*, y mayor es la influencia sobre la condición de la celda *j* a la celda *i*. No necesariamente la confianza de *i* con *j* en la misma que tiene *j* con *i* (*b(ij)* no necesariamente es igual *b(ji)*).

- Intención de propagación celular: Es la tendencia de la persona i de tomar la iniciativa de difundir la noticia/rumor. Determina cuánto cree en la noticia/rumor:

$$c(i) = y, y \in [0,1]$$

Sin considerar el grado de confianza, cuanto mayor es *c(i)*, mayor será la posibilidad de que *i* difunda la información que le llega.

Como cada persona puede ir recibiendo y transmitiendo noticias/chismes, esto hace que vaya cambiando de estados. Los tres estados en los que puede estar son: desinformado *(U)*, propagando *(S)* y desinteresado *(D)*. Un persona está en estado:

- *U* cuando no ha recibido ninguna información. Al ser expuesta a un contenido, puede convertirse en propagador *(S)* o no importarle *(D)*.

- *S* si cree en la noticia/chisme y decide difundirla a sus vecinos.
- *D* cuando no cree en la noticia y por lo tanto no difunden.

Los eventos y la transición por los distintos estados de un individuo dentro de AC se ven representados por la Figura 3.3.

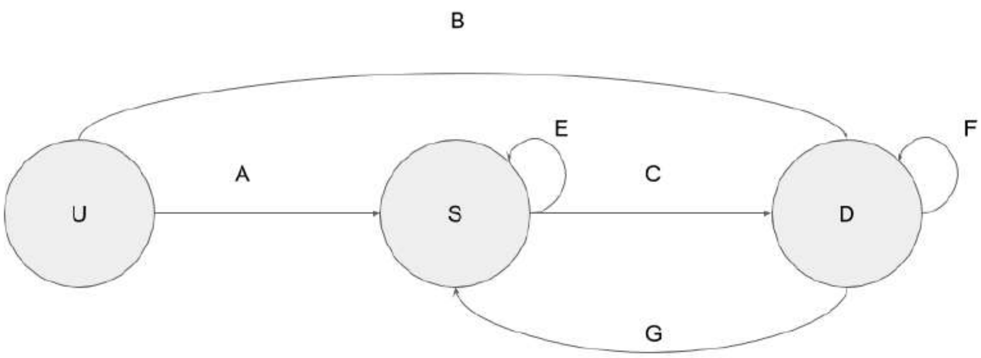


Figura 3.3 Grafo de Transiciones

donde cada una de las transiciones sucede por que la persona:

- *A*: Se entera de la noticia/chisme, cree y la transmite.
- *B*: Le llega la novedad, no cree en ella y tampoco tiene intenciones de transmitirla.
- *C*: Deja de creer en la información y decide no transmitirla más.
- *E*: Aún cree en la información y continúa su divulgación.
- *F:* Continua sin creer en la noticia/rumor.
- *G*: Comienza a creer en la veracidad de la noticia/rumor y decide comenzar a difundirla.

Todas estas transiciones dependen de la intención de propagación, el grado de confianza de la persona y desde donde le llegó la novedad, respectivamente. Por ejemplo, si la persona *j* tiene *m* vecinos *S* y *n* vecinos *D*, entonces las reglas de transición se establecen según:

- Si en el tiempo *t*, la celda central está desinformada *(U)* y algunos de sus vecinos están propagando contenido (S)($m > 0$), o tienen contenido pero no les importa difundirlo (D)($n > 0$), la celda central propagará o no la noticia en el tiempo t + 1 dependiendo de los *m* vecinos propagadores, sus respectivos grados de confianza y la susceptibilidad de persona central. Esto determina que el nodo pase a estado propagador (*S*) o desinteresado (*D*) (Transiciones U →{S, D}).

- Si la persona está propagando (S), en el siguiente instante (*t*+1) puede pasar a estar desinteresada dependiendo de su grado de susceptibilidad y la cantidad *n* de sus vecinos desinteresados y sus grados de confianza. En este caso la celda puede pasar a estado desinteresado (D) o seguir con la propagación de la noticia/rumor (S) (Transiciones S →{S, D}).
- Toda celda en estado desinformado (D), analiza sus *m* vecinos, si sus vecinos con mayor afinidad están propagando y su susceptibilidad es elevada, la celda va a pasar al estado propagación (S), de lo contrario seguirá desinteresada en la noticia/rumor (D) (Transiciones D →{D, S}).

En caso de tener varios vecinos con distintas versiones de noticias y el individuo se transforma en propagador, entonces va a considerar la versión de la mayoría de sus vecinos, en caso de no existir una versión mayoritaria, considerará aquella del vecino cuyo grado de confianza es mayor.

Por todo lo expuesto, el Prototipo de simulación para la divulgación de noticias-rumores basado en un modelo de propagación SIR, lo definimos con una AC mediante la 6-tupla $\boldsymbol{AC_{N\text{-}R}=<L,V,Q,L_0,f,r>}$ de la siguiente manera:

- **Espacio celular *(L)***: Es un arreglo bidimensional finito, con fronteras acotadas. Cada celda del autómata representa un lugar que puede ser poblado por una persona o individuo.
- **Vecindario *(V)*:** Utilizamos el vecindario de Moore (Ver Figura 3.6.b), que incluye a las 8 celdas que rodean a una celda central.
- **Estado de las celdas *(Q)*:** Una celda puede estar en uno de tres estados posibles **Q= {U, S, D}** donde:

- **Desinformado (*U*):** Las personas en estado ***U*** son aquellas que no han recibido ninguna información. Cuando son expuestas al contenido, pueden cambiar a alguno de los otros dos estados según las reglas a aplicar.

- **Propagador (*S*)**: Los individuos en estado ***S*** son aquellos que creen la noticia y la difunden a sus vecinos.

- ***Desinteresado (D):*** Las personas en estado ***D*** no creen en la noticia o rumor y por lo tanto no difunden nada.

- **Configuración inicial (L_0):** Antes que la simulación comience, se especifica el tamaño de la superficie con la que se quiere trabajar, así como la cantidad de individuos de la población y las características de cada uno: Estado, Grado de confianza, intención de propagación y versión de la noticia.
  Como en el estado inicial sólo algunos individuos conocerán la noticia/rumor, estos serán elegidos en forma aleatoria, quedando en estado *S*, los demás estarán en estado *D.*

- **Reglas de evolución (*f*):** Para determinar su estado en el siguiente instante de tiempo, cada individuo consulta su vecindad e intención de propagación. Las transiciones son:

  - ***(U→{S, D}):*** Si la celda central j está en estado *U*, se calcula

$$pj = \frac{1}{mj}\sum_{i=1}^{mj} b(j,i) * c(j)$$

  donde *mj* es la cantidad de vecinos de *j* en estado *S*, *b(j,i)* es el grado de confianza que tiene la celda *j* de la celda *i,* y *c(j)* es la intención de propagar de *j*. Según el valor de *pj*, será el nuevo estado de *j*. Así:

    - Si *pj* >= 0.5, entonces la celda *j* pasa a estado *S, **(U→S)***.
    - Si *pj* < 0.5, entonces la celda *j* pasa a estado *D, **(U→D)***.

  - ***(S→{S, D}):*** Para la celda *j*, con intención de propagación *c(j),* en estado *S* y *nj* vecinos en estado *D* con un grado de confianza de cada uno igual a *b(j,i),* se calcula

$$pj = \frac{1}{nj}\sum_{i=1}^{nj} b(j,i) * (1 - c(j))$$

Del resultado depende el estado al que se arriba en el tiempo *t +1.* De esta manera si

- Si *pj* >= 0.5, entonces la celda *j* continuará en estado *S, **(S→S)***.
- Si *pj* < 0.5, entonces la celda *j* pasa a estado *D, **(D→D)***.

- ***(D→{D, S}):*** Si la celda *j* está en estado *D*, se calcula

$$pj = \frac{1}{mj}\sum_{i=1}^{mj} b(j,i) * c(j)$$

en base a los *mj* vecinos en estado *S,* el grado de confianza que les tiene *j* y su intención de propagación. Ante el resultado se establece que:

- Si *pj* >= 0.6, entonces la celda *j* continuará en estado *S, **(D→S)***.
- Si *pj* < 0.6, entonces la celda *j* pasa a estado *D, **(D→D)***.

- **Reloj virtual (r):** El tiempo es discreto. En cada intervalo de tiempo el AC evoluciona hacia una nueva generación, las personas pueden cambiar de estado, realizando una actualización exhaustiva de sus celdas.

El autómata $AC_{N\text{-}R}$ definido previamente nos permite modelar el proceso de propagación de noticias/rumores. Realizando los ajustes y configuraciones necesarias, rápidamente puede ser adaptarlo a otra realidad. En la próxima sección vamos a analizar las características de la solución paralela de $HPC\text{-}AC_{N\text{-}R}$.

3.3. HPC-$AC_{N\text{-}R}$

Tal y como vimos en referencia, una simulación es el proceso de diseñar un modelo de un sistema real y llevar a cabo experiencias con él a fin de realizar inferencias sobre su comportamiento. Los modelos pueden ser categorizados en macroscópicos (no admiten modelar características particulares de un individuo, resultando en modelos poco flexibles) y microscópicos (permiten representar a los individuos como unidades básicas del sistema y realizar estudios más detallados de la evolución mientras transcurre el tiempo). En nuestro caso, optamos por un modelo microscópico, si bien todos los individuos se rigen por las mismas reglas, éstas se ven afectadas por características propias de cada individuo y sus vecinos.

En este trabajo se propone un sistema de simulación basado en AC y técnicas de computación de alto desempeño (HPC). El mismo tiene la capacidad de construir un ambiente dinámico, en función de ciertos parámetros, permitiendo recrear la propagación de fenómenos con propiedades de transmisión (como son la transmisión de noticias/rumores en una población). Dicho sistema desarrollado se denomina *HPC-AC$_{N\text{-}R}$*, por todas las características involucradas en su desarrollo.

En *HPC-AC$_{N\text{-}R}$*, los individuos están modelados mediante agentes, los cuales están descritos por parámetros. Estos son: estado respecto a la noticia, intención de propagación, grado de confianza que le tiene a cada uno de sus vecinos y versión de la noticia. En la Figura 3.4 podemos ver el ambiente a simular.

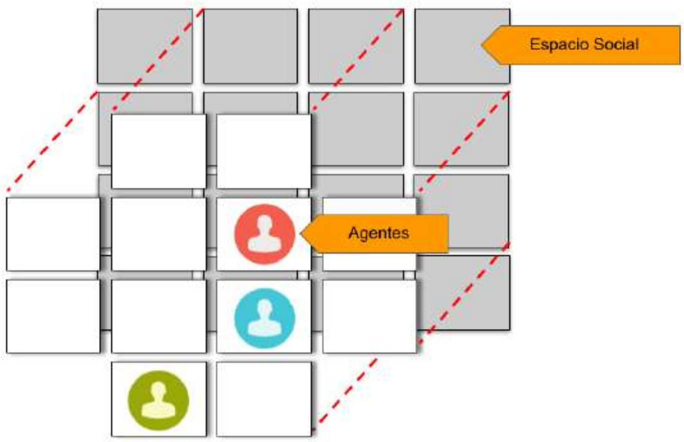


Figura 3.4. Ambiente a Simular

De acuerdo con lo expuesto, el espacio social se describe mediante un AC extendido, especificado por una 6-tupla de la forma *(L, V, Q, L_0, f, r)*, tal y como se definió en la sección 2.2.1. Las reglas de evolución de AC están destinadas a modelar la propagación de un sistema de difusión utilizando como base el modelo epidemiológico SIR.

Es importante mencionar, que la propagación está pura y exclusivamente dada de manera estocástica, aumentando dicha probabilidad en función de la cantidad de agentes conocedores o no de la noticia/rumor en el vecindario.

Cada individuo interactúa con los demás a través de su vecindario, a partir de su celda o posición en el espacio social. Así, podemos establecer que cada celda del ambiente $c_i \in L$, puede contener como máximo un agente.

En las siguientes secciones se detallan las características principales de diseño e implementación del prototipo paralelo.

3.3.1 HPC-AC$_{N\text{-}R}$: Aspectos de Diseño

Nuestro modelo representa los cambios dinámicos, a través del tiempo, de cómo una noticia/rumor se va difundiendo en una población con características específicas y en función de cambios estocásticos, los que sólo asumen valores discretos a lo largo del desarrollo de la simulación. Todo esto hace que HPC-AC$_{N\text{-}R}$ sea clasificado como: dinámico, discreto y estocástico.

La población está inmersa en un espacio social representado por una grilla bidimensional de tamaño configurable, la cual contiene un conjunto de celdas donde cada una puede a lo más contener una persona. Si dos celdas contiguas tienen una persona, esto indica que éstas están en contacto directo y, por lo tanto, pueden transmitir la noticia/rumor. La probabilidad de transmisión depende entre otras cosas de la cantidad de vecinos del individuo susceptible a enterarse, de su intención de propagación y del grado de confianza que tiene con los vecinos conocedores de la novedad.

Como se mencionó anteriormente, cada persona es representada por distintas propiedades; ellas son: el estado, el grado de confianza con cada uno de sus vecinos, su intención de propagación y la versión de la noticia que manipula, si es verdadera o falsa. El estado de una persona puede ser: Desinformado, Propagador o Desinteresado.

En un inicio, el ambiente sólo tiene personas desinformadas y propagadoras (quienes conocen una versión de la noticia/rumor y actúan como difusores). HPC-AC$_{N\text{-}R}$ muestra los cambios en el sistema, avanzando a pasos discretos de tiempo, donde el lapso de tiempo de cada paso es configurable. Cada paso de la simulación implica una actualización completa del espacio celular. Este comportamiento continúa hasta que todas las personas están en estado Desinteresado o hayan transcurrido *X* días (*X* es un parámetro configurable). Ambos eventos marcan el fin de la simulación.

Para observar la evolución y propagación de una noticia, el sistema cuenta con una interfaz gráfica simple, permitiendo realizar algún análisis para fines educativos (Ver Figura 3.5: Celda Blanca representa al vacío, Verde: Desinformado, Celeste: Desinteresado y Rojo: Propagador).

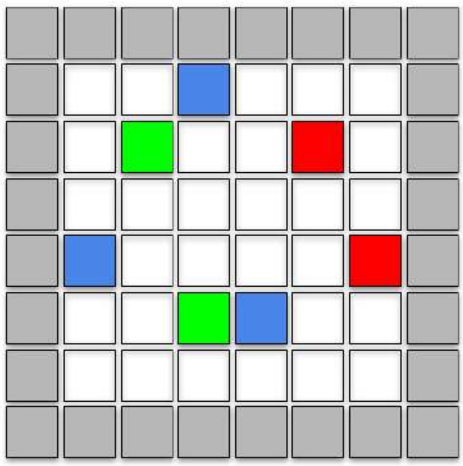


Figura 3.5. Representación del Espacio Social

Cada paso del tiempo supone una actualización completa del AC, siendo necesario asegurar la atomicidad y el determinismo en la simulación por cada transcurso del reloj.

En la Figura 3.6 se muestra la arquitectura del sistema, el cual se desarrolla en fases y en cada una de ellas se realizan las siguientes tareas:

- **Fase de actualización de Estado**: Divide el espacio social entre los procesos y en paralelo, cada uno calcula el estado siguiente de cada celda que le corresponde.
- **Fase de Consistencia**: Reúne todos los estados calculados por los distintos procesos y controla la consistencia del sistema, dejando las estructuras de datos actualizadas para iniciar el siguiente paso de la simulación.

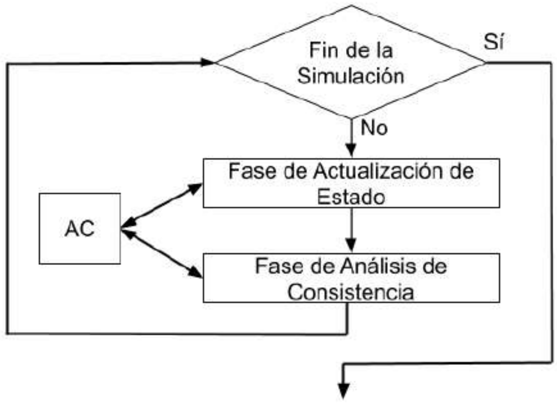


Figura 3.6. Fases de la Simulación

La actualización de las celdas en cada paso de tiempo se efectúa de manera exhaustiva y en un orden determinado. Esto podría ser realizado de manera exhaustiva y aleatoria, lo cual implica llevar un control extra de las actualizaciones realizadas.

En la próxima sección se detallan diferentes características de la implementación de HPC-AC$_{N\text{-}R}$.

3.3.2. HPC-AC$_{N\text{-}R}$: Aspectos de Implementación

Si bien en este trabajo hemos considerado la difusión de una noticia/rumor, el sistema está desarrollado para trabajar con cualquier fenómeno o actividad de difusión: virus informático, fluidos, fuego, etc. Por ello, lo primero a definir es el modelo matemático de propagación. Como mencionamos en el capítulo anterior, el modelo epidemiológico SIR permite determinar cómo se propagan las enfermedades en una población, esta versión básica resulta suficiente para representar la realidad y las diferentes etapas por la que pasa una persona desde el momento de no conocer nada a enterarse de una noticia/rumor y difundirlo o no.

El pseudocódigo de la Figura 3.7 detalla nuestro proceso de simulación utilizando la difusión de noticia/rumor como base; la misma cuenta con dos puntos de salida: se alcanzaron los días máximos para la simulación o todas las personas están desinteresadas en divulgar la noticia. Cada una de las fases descriptas en la sección anterior se resuelve mediante tareas paralelas, las cuales, al mismo tiempo, recorren y actualizan el estado de las personas. Cada fase depende de la fase anterior y en cada una se definen varias tareas paralelas.

Respecto a las técnicas de computación de alto desempeño, el planteo de las tareas se realizó siguiendo un modelo de paralelismo de datos, donde todos los procesos realizan lo mismo. En este caso uno de ellos se destaca, denominado Líder, el cual es el responsable de la tarea de particionar los datos, enviarlos a los demás procesos, recibir de estos y armar el nuevo AC.

```
Si es el proceso Líder
        Lee el estado inicial
        Divide los datos
        Envía cada partición al proceso correspondiente.
Mientras no es el Final de la Simulación
         Si (no es el proceso Líder)
                 Recibe la partición desde el Líder
         Calcula el nuevo estado de cada celda en su partición
         Si (no es el proceso Líder)
                 Envía la nueva partición calculada al Líder
         Sino
                 Recibe las particiones de cada uno de los otros procesos
                 Arma el nuevo AC
                 Divide los datos
                 Envía cada partición al proceso correspondiente.
                 Recolección de estadísticos.
Mostrar resultados estadísticos
```

Figura 3.7. Pseudocódigo de HPC-$AC_{N\text{-}R}$

Con esto en mente y teniendo en cuenta la definición del autómata $AC_{N\text{-}R}$., la sextupla *(L, V, Q, L_0, f, r)* se define como:

- Espacio celular (*L*): Es un arreglo bidimensional finito (matriz) con frontera abierta (Caparrini, 2016). Cada celda representa un lugar a ser ocupado por una persona. La matriz no representa un espacio físico en sí, sino más bien una abstracción, donde dos celdas adyacentes ocupadas representan una interacción directa o indirecta de los individuos (por ejemplo, dos personas conversando tomadas del pasamanos de un colectivo). A este espacio lo denominamos espacio social.
- Vecindario (*V*): En este trabajo utilizamos el vecindario de Moore; cada celda tiene como vecinas a las 8 celdas que la rodean. De esta manera se maximiza la capacidad de interacción de cada individuo. Es posible la utilización de otros tipos de vecindarios. A futuro, se podrían realizar nuevos estudios para determinar el impacto de diferentes variantes.

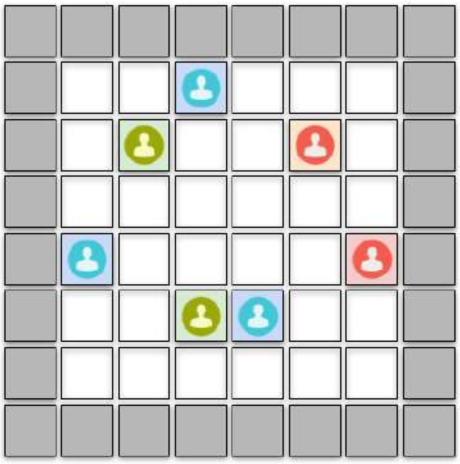


Figura 3.8. Espacio Social

- Estado de las celdas (*Q*): Como nos basamos en el modelo SIR y consideramos que las fronteras del AC son abiertas, una celda puede estar en uno de los estados del siguiente conjunto *Q* = *{B, F, U, S, D}*. A fines prácticos, y como nos basamos en el modelo SIR, cuando una celda está ocupada con una persona, el estado de la celda se transforma en el estado de la persona. Así los estados B y F se corresponden al estado de una celda, mientras que U, S y D son del estado de una persona y significa que la celda está ocupada. Cada estado significa:
    - *B:* Límite del Autómata
    - *F:* Celda Libre.
    - *U:* Persona susceptible, no conoce la noticia/rumor.
    - *S:* Persona conocedora de la noticia/rumor y dispuesto a difundirlo.
    - *D:* Persona conocedora de la noticia/rumor, sin intenciones de difundirlo.

En la Figura 3.9 se puede ver un espacio social con celdas en distintos estados.

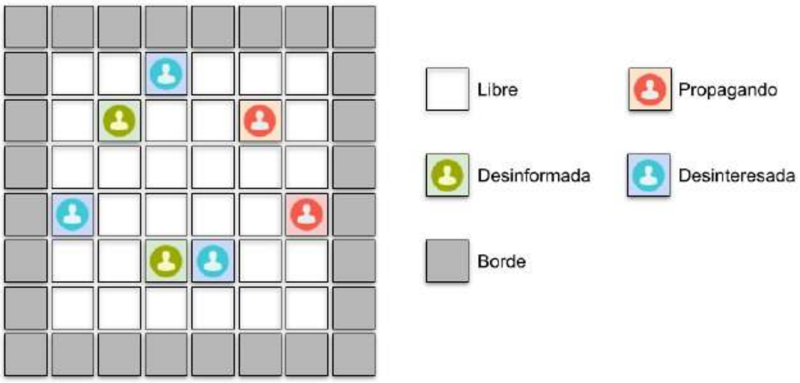


Figura 3.9. Estados de las Celdas

- Configuración inicial (L_0): antes que la simulación comience, se especifica toda la información relevante a cómo va a estar distribuida la población en la matriz; L tiene una densidad de población y porcentaje de conocedores de la noticia inicial definido.
- Reglas de evolución (*f*): Dividimos las reglas de evolución en dos grupos:
  - Reglas sobre el espacio: una celda en estado *B* (considerando bordes abiertos) no cambiará su estado en ningún momento a lo largo de la simulación.
  - Reglas acerca de la propagación de la novedad: Son las definidas en la Sección 3.2.2. Un vecino en estado *B*, se lo considera celda libre.
- Reloj virtual (*r*): debido a la realidad en la que vivimos y la cantidad de medios a través de los cuales nos podemos comunicar, el paso de tiempo en la simulación es configurable. Para las ejecuciones, tomamos un paso de una hora.

Por último, pero no menos importante, debemos abocarnos a la computación de alto desempeño. Por su naturaleza, los AC son altamente paralelizables y en consecuencia se puede hacer uso de diferentes técnicas de HPC para mejorar su rendimiento. En particular, para nuestro trabajo aplicamos estas técnicas para arquitecturas de memoria distribuida.

En términos generales, cada AC debe actualizar todas sus celdas para llegar al próximo paso. En un instante de tiempo, cada celda depende de sí misma y de sus vecinos, permitiendo esto procesarlas al mismo tiempo y en forma independiente, generando un nuevo AC. Cómo trabajamos en una arquitectura de memoria distribuida, la actualización de una celda no afecta a ninguna adyacente.

Dada la dependencia existente entre las dos etapas del sistema, las mismas no pueden ser resueltas simultáneamente, son implícitamente secuenciales.

El simulador fue desarrollado íntegramente en Python; por como González Duque (2010) menciona en su libro, Python es un lenguaje con sintaxis simple, clara y sencilla (cercana al lenguaje natural); el tipo de datos es dinámico, posee gran variedad de bibliotecas disponibles y potencia; además de licencia libre. Estos aspectos, entre otros, hacen que el desarrollo de una aplicación en Python sea sencillo y prolijo.

Soporta programación multithreading, permitiendo la utilización de MPI (biblioteca de Memoria Distribuida) para el desarrollo actual, y OpenMP (biblioteca de Memoria Compartida) a fin de explorar nuevas capacidades en trabajos futuros.

En las siguientes secciones atenderemos aspectos puntuales de la implementación.

3.3.2.1. Estructuras de Datos

Para establecer un sistema dinámico, el lenguaje nos brinda distintas herramientas para lograr simular el AC, solo conociendo su dimensiones, los conceptos de AC unidimensional y bidimensional podemos adaptarlos al de una lista convencional (Figura 3.10).

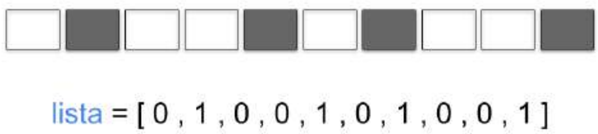


Figura 3.10. AC unidimensional con población N = 10 células (gráficamente y en líneas de código)

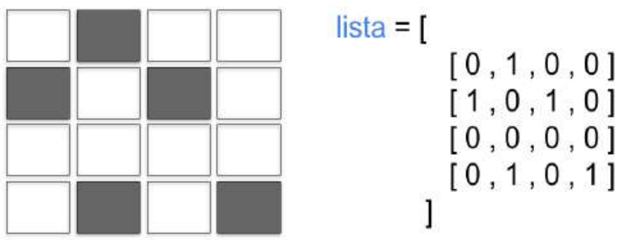


Figura 3.11. AC bidimensional con población N = 16 células (gráficamente y en líneas de código)

Para nuestra simulación, se decidió estructurar la lattice sobre listas de lista o listas anidadas con una población o espacio celular de tamaño $N = m*n$ células, donde m y n pueden no ser iguales y deben ser mayores a cero (Figura 3.11).

Con la incorporación de sublistas en el interior de una lista raíz, logramos imitar la conformación de una matriz de m x n, donde las m filas son representadas por las sublistas y las n columnas por los elementos dentro de la sublistas de igual tamaño.

Para las celdas fue necesario utilizar una estructura de datos perteneciente al lenguaje denominada "diccionarios". Los diccionarios nos permiten almacenar cualquier tipo de valor como enteros, cadenas, listas e incluso otras funciones además de identificar cada elemento por una clave.

3.3.2.2. Programación de Alto Rendimiento

Para aplicar técnicas de alto rendimiento se utilizó la biblioteca de MPI (MPI4py). Inicialmente se desarrolló el sistema de manera secuencial, obteniendo todos los resultados y validando el funcionamiento del mismo. Una vez finalizada dicha etapa, se procedió a paralelizar el código desarrollado. El trabajo se planteó haciendo una división de tareas según la descomposición de dominios o los datos, para ello cada tarea se encarga de procesar una porción de L. En este caso la división se realizó en sectores, los cuales se corresponden con las filas de la matriz. El siguiente pseudocódigo (Figura 3.12) muestra la sección paralela donde se analiza el próximo estado de la celda.

En paralelo cada proceso:

```
for r in range( 1, len( partition ) - 1):
  for c in range( 0, len( partition[ r ] ) ):
        result = evaluate( game,  r,  c )
```

Figura 3.12. Análisis el próximo Estado de la Celda

El conjunto de tareas involucradas serán realizadas por p de procesos definidos al momento de la ejecución.

La descomposición de L nos permite fraccionar el AC en porciones a ser distribuidas entre los *p* procesos. Las particiones son confeccionada en dos etapas:

- Primera etapa: Las filas de L se dividen en función de la cantidad de tareas *p* de manera de lograr el equilibrio de carga ideal.

  A continuación, en la Figura 3.13, un ejemplo que involucra una matriz pequeña de 5x4, siendo f = 4, donde la equidad sólo se dará en los casos donde la cantidad de procesos sea p = 2 o p = 4.

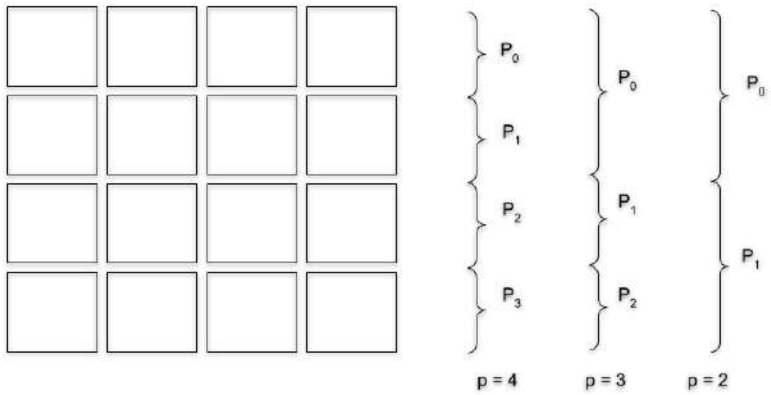
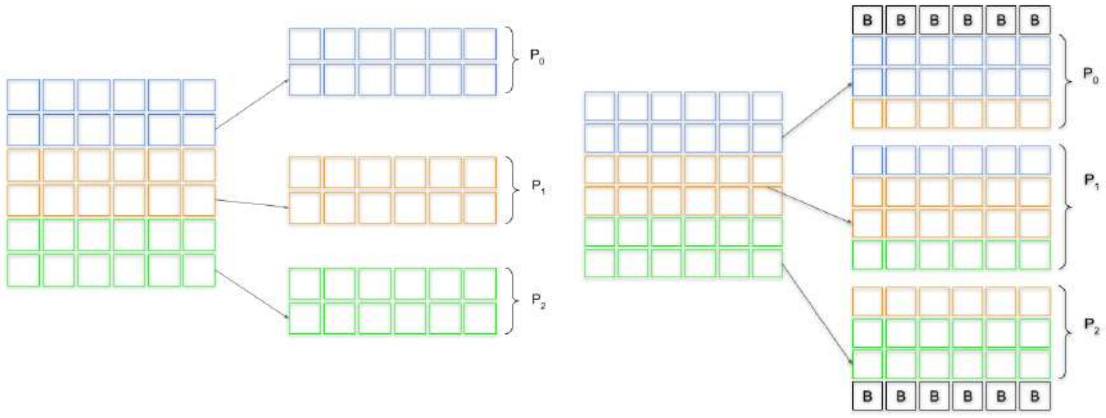


Figura 3.13. Descomposición de Dominio en Relación al Número de Filas y al de Procesos

  Obsérvese en el ejemplo, que cuando el número de proceso *p* es igual a 3, al no cumplir la condición de divisibilidad se presenta una sobrecarga para el proceso cero (P_0). Si fueran 5 las filas del AC, entonces P_0 y P_1 estarían a cargo de 2 filas y P_2 de sólo 1.

- Segunda etapa: Ya determinadas las filas a procesar por cada tarea, considerando que la vecindad escogida es conformada por 8 vecinos (vecindad de Moore) y que los procesos analizarán su partición en forma independiente es necesario que cuenten con dichas celdas vecinas. Esto permite analizar la vecindad sin recurrir al intercambio de mensajes entre los *p* procesos. En la figura 3.14(a) se muestra cómo se dividen las filas en los 3 procesos. En la figura 3.14(b) se muestran los datos necesarios para computar todas las celdas asignadas a cada proceso. Para los procesos 0 y *p-1* (zonas superior e inferior del autómata) la fila superior (p_0) y la inferior (p_2) corresponden a las celdas de la frontera.



(a) (b)

Figura 3.14. Representación de Porciones con Filas Auxiliares

Tanto la división como la combinación de las partes de *L* son realizadas por el proceso p_0. Al momento de reunir las fracciones para obtener el nuevo AC, las filas adicionales no son enviadas al proceso unificador.

Cuando los datos están disponibles en cada proceso, se efectúa el análisis y actualización de cada parte. Las celdas modificadas serán las correspondientes a su porción de datos, las filas extras sólo son consultadas por su estado. Una vez realizado su trabajo, todos los procesos envían a p_0, quien se encarga de ensamblar *L*.

3.4 Pruebas y Análisis del Desempeño de Simulación

En esta sección se presentan los experimentos realizados y el análisis de los resultados de HPC-$AC_{N\text{-}R}$. Con el objetivo de modelar diversas realidades, se consideraron escenarios de distintas características y dimensiones. Esto permitió, además de analizar el comportamiento del simulador, comprobar su desempeño para diferentes tamaños del problema.

Los resultados presentados fueron obtenidos utilizando la técnica de replicación de ejecuciones, donde una serie de simulaciones completamente separadas e independientes son llevadas a cabo con los mismos parámetros. Así, cada ejecución representa una observación individual, pudiéndose obtener una media de todas las ejecuciones ya que tienden a estar normalmente distribuidas en virtud del teorema central del límite[3]. No existe un número determinado de ejecuciones para asegurar la aleatoriedad de la muestra, pero obviamente, mientras más, es mejor. En nuestro caso, para cada configuración trabajada, se realizaron 10 repeticiones.

3.4.1 Escenarios y Casos de Prueba

Las pruebas de laboratorio para los diversos escenarios planteados fueron realizadas sobre un Cluster, propiedad de la Universidad Nacional de San Luis, al que obtuvimos acceso en forma de colaboración al Proyecto PIDAC (Resolución CS N°432-18): “Cómputo de Altas Prestaciones aplicado en la Solución de Grandes Problemas”. El mismo cuenta con la siguientes características:

[3] El teorema del límite central o teorema central del límite indica que, en condiciones muy generales, si S_n es la suma de n variables aleatorias independientes y de varianza no nula pero finita, entonces la función de distribución de S_n «se aproxima bien» a una distribución normal

ClusterMulticore	Descripción
1 CPUs (Front-end)	Procesador: Intel Core 2 Duo E8200 2.66 GHz Memoria : 4GB DDR3 1333Mz Disco Rígido: 500 Gb SATA Disco Rígido: 2 TB SATA Placa Madre : Asus P5KPL-AM Placa de video: GeForce 9400 GT de 512 MB
1 CPUsMulticore Dell PowerEdge R815 (nodo de ejecución)	Procesador: 4x AMD Opteron 6128, 2.0GHz, 8C, 4M L2/12M L3, 1333Mhz Memoria: 64GB Memory (16x4GB), 1333MHz, Disco Rígido: 500GB 7.2K RPM Near-Line SAS 6 Gbps 2.5in HotPlug Hard Drive Fuente de alimentación: 1100 Watt Redundant Power Supply
3 CPUsMulticore Dell PowerEdge R815 (nodos de ejecución)	Procesador: 2x AMD Opteron 6272, 2.1GHz, 16C, 16M L2/16M L3, 1333Mhz Memoria: 64GB Memory (16x4GB), 1333MHz, Disco Rígido: 500GB 7.2K RPM Near-Line SAS 6Gbps 2.5in HotPlug Hard Drive Fuente de alimentación: 1100 Watt Redundant Power Supply
1 Switch	Linksys SRW 2024 de 24 bocas
Sistema Operativo	Debian 8 (Jessie) de 64 bits

Tabla 1. Características del Cluster

Los escenarios se pensaron considerando diferentes espacios sociales. Ellos se definieron como una combinación de los siguiente parámetros:

- **Tamaño de grilla:** Tamaño del autómata.
- **Muestra:** Cantidad de lugares ocupados en la grilla, o sea la población.
- **Infección:** Número de personas o individuos que están en estado de propagación o conocen una versión de la noticia/rumor.

Los días de simulación son 7 con una actualización del reloj cada 2 horas. La siguiente tabla (Tabla 2) muestra los valores utilizados en la definición de los 10 escenarios definidos, además del rango de variación de los parámetros de cada individuo: *Grado de Propagación* y de *Confianza con cada vecino*.

Parámetro	Valor
Tamaño Grilla	100 x 100; 500 x 500; 2.000 x 2.000
Muestra (%)	25%; 50%; 80%
Infección (%)	0.1%; 1%
Versiones de noticias	Falso, Verídico
Intervalo grados de propagación	$0.3 \leq grado \leq 1$
Intervalo grados de relación vecinal	$0.3 \leq grado \leq 1$
Iteraciones (2hs)	84 (1 sem)

Tabla 2. Parámetros de Ejecución

En la Tabla 3 se muestran las características de los 10 escenarios definidos según la combinación de los parámetros anteriores. Algunas combinaciones no fueron utilizadas por carecer de sentido.

<table>
<tr><th>Escenarios</th><th>Dimensiones</th><th>Muestras</th><th>Grados de infecciones</th></tr>
<tr><td>#1</td><td rowspan="2">100 x 100</td><td rowspan="4">50%</td><td>0.1%</td></tr>
<tr><td>#2</td><td>1%</td></tr>
<tr><td>#3</td><td rowspan="2">500 x 500</td><td>0.1%</td></tr>
<tr><td>#4</td><td>1%</td></tr>
<tr><td>#5</td><td rowspan="4">2.000 x 2.000</td><td rowspan="2">25%</td><td>0.1%</td></tr>
<tr><td>#6</td><td>1%</td></tr>
<tr><td>#7</td><td rowspan="2">50%</td><td>0.1%</td></tr>
<tr><td>#8</td><td>1%</td></tr>
</table>

#9		80%	0.1%
#10			1%

Tabla 3. Configuración de Escenarios

3.4.2 Análisis de Resultados de HPC-AC$_{N-R}$

De los resultados obtenidos podemos realizar dos tipos de análisis, uno relacionado al desempeño del simulador y otro al comportamiento de la propagación de noticias. Para el primer caso se evaluó la performance de HPC-AC$_{N-R}$ considerando 1, 2, 4, 8, 16 y 32 procesos. Las métricas evaluadas son la aceleración (S) y la eficiencia(E) (Pacheco, 2011).

En la Tabla 4 se especifican los tiempos en segundos que demora la simulación para cada uno de los escenarios y la cantidad de procesos (en todos los casos, siempre coincidió con la cantidad de procesadores utilizados). En la segunda columna se detalla el tiempo secuencial involucrado en resolver la aplicación en secuencial. En la Figura 3.15 se pueden apreciar dichos valores gráficamente.

Procesos→ Escenarios↓	1	2	4	8	16	32
#1	54.832	30.142	17.040	10.771	7.552	6.965
#2	53.938	30.556	17.652	10.422	7.409	6.332
#3	1516.353	868.323	491.283	341.354	252.021	165.542
#4	1521.299	870.376	491.180	304.685	218.432	166.772
#5	19920.131	11695.062	6811.582	4434.129	3282.600	3033.574
#6	20064.811	11651.798	6812.622	4502.337	3091.966	2665.864
#7	25184.284	14497.012	8429.315	5314.603	3771.615	3128.786

#8	24990.113	14643.480	8222.338	5273.666	3933.778	3265.576
#9	30793.007	17626.809	10187.207	6396.886	5248.616	3703.135
#10	31025.596	18011.203	10235.011	6577.890	5341.049	3970.963

Tabla 4. Tiempo promedio en segundos de HPC-ACN-R

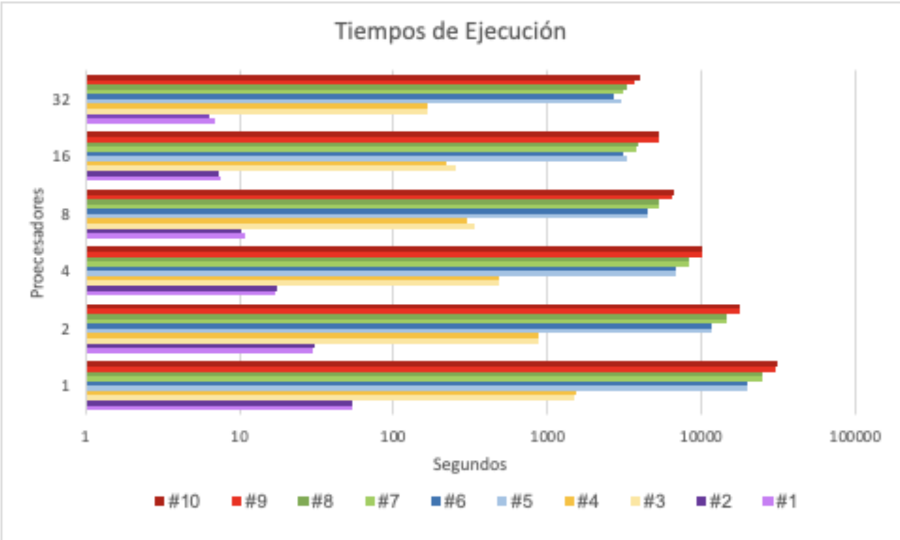


Figura 3.15 Tiempo promedio de HPC-ACN-R expresado en escala logarítmica

Como puede observarse el número de infectados iniciales incide muy poco en el tiempo de ejecución, sí tiene influencia la cantidad de población, los escenarios más poblados tardan más en procesarse que los menos poblados.

En las Figuras 3.16 y 3.17 se muestra la aceleración y eficiencia lograda por la implementación paralela.

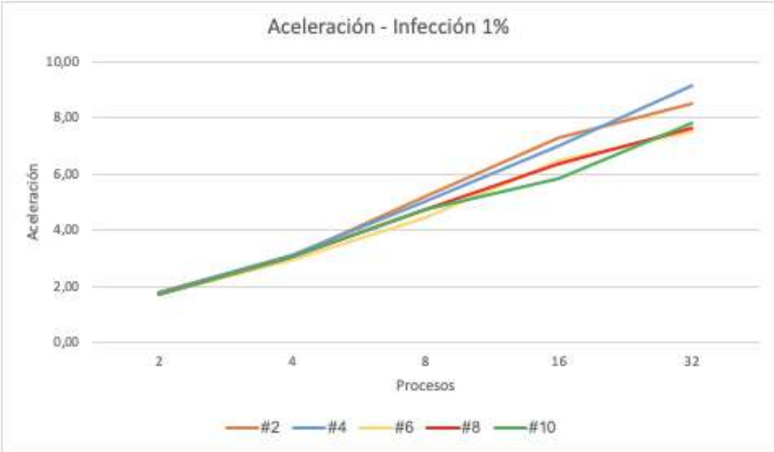


Figura 3.16 Aceleración de HPC-ACN-R para una Población inicial Difusora del 1%

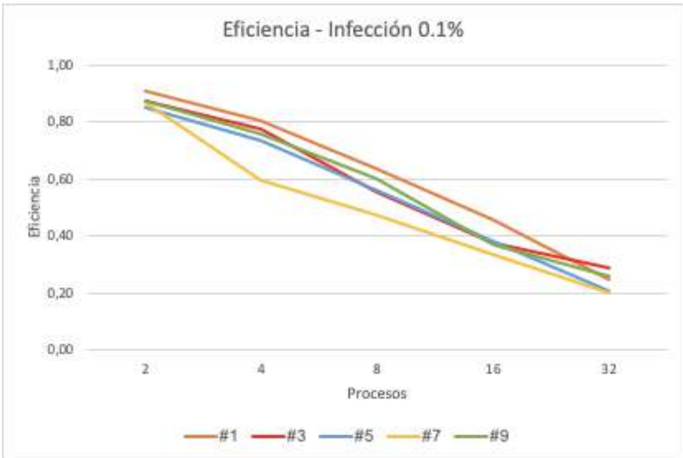


Figura 3.17 Eficiencia de HPC-ACN-R para una Población inicial Difusora del 0,1%.

Se puede observar en la Figura 3.16 la aceleración obtenida, la cual fue mejorando a medida que se incrementa la cantidad de procesos. El comportamiento es similar en todos los escenarios, independientemente del tamaño de la población. La Figura 3.17 nos muestra la eficiencia lograda, en ella se puede observar, que si bien la aceleración es mayor a medida que el número de procesos crece, existe una subutilización de los mismos; la mejor eficiencia se logra para dos proceso, a partir de allí empieza a decrecer. Este comportamiento puede obedecer a múltiples factores, uno de ellos es la influencia de las comunicaciones en el cómputo, las cuales se incrementan al aumentar el número de procesos.

Además del análisis de desempeño del simulador paralelo, otro aspecto importante es analizar el comportamiento de los ambientes sociales en diferentes situaciones. Uno de los estudios realizados fue determinar qué se difunde más rápido, si una noticia o un rumor. En la Figura 3.18 se puede ver como para una superficie de 2000 x 2000, distintas densidades de población y de infección inicial, siempre la noticia falsa se distribuye y llega a más personas que la verdadera. Lo mismo pasa para distintas superficies e igual densidad poblacional y grado de infección, Figura 3.19 y 3.20 respectivamente. Las escalas en todas las gráficas son logarítmicas.

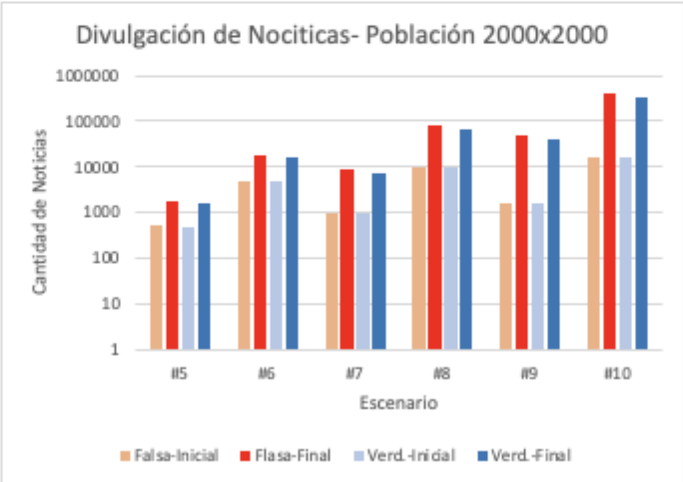


Figura 3.18. Cantidad de Noticias Verdaderas y Falsas difundidas en una superficie de 2000x2000 con distintas densidades poblacionales

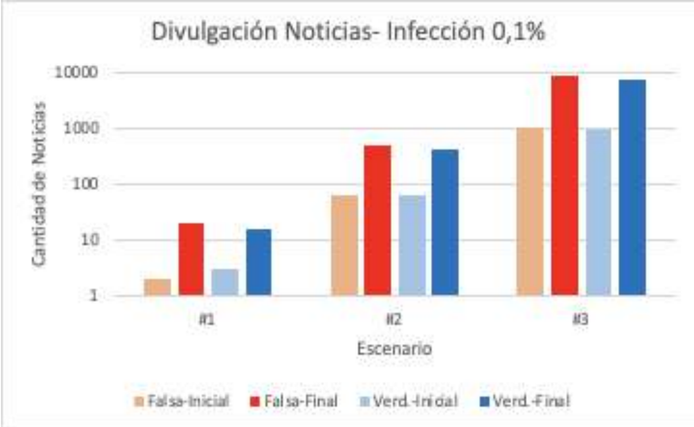


Figura 3.19. Cantidad de Noticias Verdaderas y Falsas difundidas en distintas superficies-50% de densidad poblacional - 0,1% de infección inicial

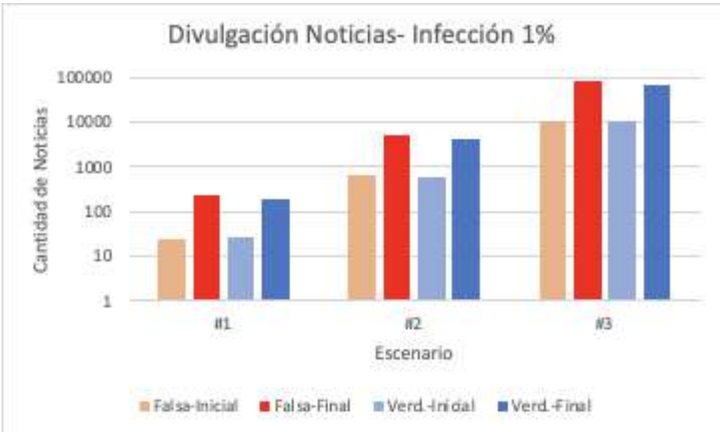


Figura 3.20. Cantidad de Noticias Verdaderas y Falsas difundidas en distintas superficies-50% de densidad poblacional - 1% de infección inicial

Estos resultados nos permiten comprobar que la hipótesis planteada es válida, las noticias falsas se distribuyen más rápidamente que las verdaderas. Esto abre un gran número de interrogantes, por ejemplo ¿A qué obedece este comportamiento? ¿Cuál es el grado de morbo en la difusión? ¿Es posible validar estos resultados? Todas estos cuestionamientos y muchos más hacen que este trabajo sea sólo el inicio de una larga e interesante investigación.

CAPÍTULO 4

Conclusiones y Trabajos Futuros

Para finalizar, en este capítulo se plantean las conclusiones en base al objetivo e hipótesis planteados y las apreciaciones de los resultados obtenidos. Además se proponen algunas recomendaciones para la continua mejora del modelo HPC-AC_{N-R} y futuras líneas de investigación.

4.1. Conclusiones

En este trabajo se plantea el objetivo de diseñar e implementar un prototipo utilizando AC y técnicas de HPC para simular el comportamiento de una sociedad frente a la dispersión de dos versiones de una misma noticia o rumor. En función de ello se definió el AC, especificando los estados y reglas de transición y se desarrollaron dos propuestas: La primera, utilizando programación secuencial y la segunda, con programación paralela utilizando la biblioteca de pasaje de mensajes MPI. Ambas soluciones han sido puestas a prueba, verificando su funcionamiento, validando los resultados generados y analizando su performance en diferentes escenarios.

La combinación del funcionamiento del modelo de AC y las técnicas de HPC en un ambiente formado por un cluster de computadoras han mostrado resultados prometedores, permitiendo resolver aplicaciones de alta demanda computacional, con un muy buen desempeño. Esto da la posibilidad de tratar problemas a gran escala, los cuales se caracterizan entre otros aspectos, por requerir mucho tiempo de ejecución, o incluso en algunos casos ser imposibles de resolver con los recursos disponibles.

Los resultados de rendimiento y comportamiento obtenidos indican que el prototipo paralelo es una buena herramienta de análisis. En un tiempo reducido, podría ser de utilidad para ayudar a comprender los fenómenos de propagación de distintas versiones de noticias en distintos contextos de comunicación. Nos permitiría evaluar y determinar en un corto tiempo estrategias de difusión de noticias por ejemplo en campañas políticas, comerciales, etc.

Al mismo tiempo presenta la ventaja de la portabilidad y una escalabilidad aceptable, principalmente al incrementar el tamaño del problema a resolver. Además constituye una herramienta útil para su aplicación en diferentes ámbitos, como así también en distintas problemáticas de difusión.

4.2. Publicaciones

El trabajo acá desarrollado fue difundido en los siguientes congresos con referato:

- Barrionuevo, Mercedes, Escalante, Julián, Lopresti, Mariela, Lucero, Maximiliano, Miranda, Natalia Carolina, Murazzo, María Antonia, Piccoli, María Fabiana. Solución de grandes problemas aplicando HPC multi-tecnología. XXII Workshop de Investigadores en Ciencias de la Computación (WICC 2020, El Calafate, Santa Cruz). Red de Universidades con Carreras en Informática. Universidad Nacional de la Patagonia Austral. ISBN: 978-987-3714-82-5. Pp 776-781. Mayo 2020.

- Escalante, J., Miranda N., Piccoli F. (2020). HPC-ACN-R: Un Simulador Paralelo para el Análisis de la difusión de Noticias/Rumores. 8º Congreso Nacional de Ingeniería Informática y Sistemas de Información (CoNaIISI). UTN Reg. San Francisco. Noviembre 2020. En prensa.

4.2 Trabajos Futuros

Este trabajo deja varias líneas de investigación abiertas, entre ellas se encuentra la evaluación del prototipo aplicando otros paradigmas de programación HPC, como es el caso de memoria compartida o en otras arquitecturas como las GPU, e incluso el paralelismo híbrido o lenguajes paralelos específicos como por ejemplo Go.

Respecto al modelo de propagación de noticias/rumores, si bien resultó útil para entender la dinámica de distribución de contenidos en una población, existen números tópicos a mejorar. Por ejemplo, continuando en la aproximación del modelo, indagar en qué otros factores propios de las relaciones sociales y de influencia podemos adoptar con el fin de ajustar nuestra investigación a otras realidades como por ejemplo incorporar mayor descripción a los agentes (sexo, edad, ocupción, entre otros), reglas de movimiento (realidad donde las personas pueda cambiar de ubicación), condiciones de aislamiento de personas (sin contacto estrecho con sus vecinos), de “sobreinformación”, definición dinámica de vecindario, o mecanismo de ajuste para las probabilidades de cambio y detección de nuevos patrones mediante técnicas de Inteligencia Artificial, entre otros.

Otro aspecto importante es la validación de los resultados obtenidos mediante la comparación con el comportamiento de la población en situaciones reales.

Referencias Bibliográficas

- Piccoli, María Fabiana (2011). Computación de alto desempeño en GPU. La Plata, Argentina. Editorial de la Universidad Nacional de La Plata (EDULP).
  Recuperado de: Vínculo 1

- Almeida, Francisco; Giménez Domingo; Mantas, José M.; Vidal, Antonio M. (2008). Introducción a la Programación paralela. Madrid, España. Paraninfo.

- Joyanes Aguilar, Luis (2008). Fundamentos de Programación: Algoritmos, estructura de datos y objetos. Madrid, España. McGRAW-HILL.

- Pacheco, Peter S. (2011). An Introduction to Parallel Programming. Burlington, USA. Elsevier Inc.

- Grama, Ananth; Gupta, Anshul; Karypis, George; Kumar, Vipin (2003). Introduction to Parallel Computing. Pearson Education.

- Piccoli, María Fabiana; Printista, Alicia Marcela (2001). Técnicas estándares para paralelismo anidados de datos. Repositorio Institucional de la UNLP.
  Recuperado de: Vínculo 2

- Message Passing Interface Forum (2015). MPI: A Message-Passing Interface Standard Version 3.1.
  Recuperado de: Vínculo 3

- Alonso, José Miguel (1997), Programación de aplicaciones paralelas con MPI (Message Passing Interface).
  Recuperado de: Vínculo 4

- González Duque, Raúl (2010). Python para todos. Recuperado de: Vínculo 5

- Wachenchauzer, Rosita; Manterola, Margarita; Curia, Maximiliano; Medrano, Marcos; Paez Nicolás (2011). Algoritmos y Programación I Con lenguaje Python. Recuperado de: Vínculo 6

- Reyes Gómez, David Alejandro (2011). Descripción y aplicaciones de los autómatas celulares. Departamento de Aplicación de Microcomputadoras, Universidad Autónoma de Puebla. Recuperado de: Vínculo 7

- Caligaris Marta G.; Rodríguez,Georgina B. (2010). Simulaciones computacionales: autómatas celulares. Recuperado de: Vínculo 8

- Gómez Ramírez, Isabel Cristina (2015). Modelo para el análisis y la simulación del proceso de difusión del marketing viral. Recuperado de: Vínculo 9

- Delmazo, Caroline; Valente, Jonas C.L. (2018). Fake news en las redes sociales online: propagación y reacciones a la desinformación en la búsqueda por clics. Recuperado de: Vínculo 10

- Espinoza, J.; Santero, G.; Franceschi, M. J.; Giménez, J.; Escalante, M. ; Caneva J. y otros (2019). Fake news, pos-vedad, trolls. Repositorio Institucional de la UNLP. Recuperado de: Vínculo 11

- Brito do Nascimento, Francisca Joamila; Takeshi Seo, Rodrigo (2017). Fake news cellular automata model. Recuperado de: Vínculo 12

- Xiaoxia Zhu, Jiajia Hao, Yuhe Shen Tuo Liu, Mengmeng Liu (2018). Diffusion of false information during public crises: Analysis based on the cellular automaton method. Recuperado de: Vínculo 13

- Rinaldi, Pablo R. (2011). Modelos de Autómatas Celulares sobre Unidades de Procesamiento Gráfico de Alta Performance. Recuperado de: Vínculo 14

- Palach, Jan (2014). Parallel Programming with Python. Packt Publishing. Birmingham, Reino Unido.

- Zaccone, Giancarlo (2015). Python Parallel Programming Cookbook. Packt Publishing. Birmingham, Reino Unido.

- Dalcin, Lisandro (2019). MPI for Python Release 3.1.0a0.
Recuperado de: Vínculo 15

- Prasad, Sushil K; Gupta, Anshul; Rosenberg, Arnold L; Sussman, Alan; Weems, Charles C. (2015). Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses. Wyman Street, Waltham, MA 02451 USA. Elsevier Inc

- López Salinas, Ana Miriam (2011). Introducción a la Vida Artificial y Autómatas Celulares. Departamento de Aplicación de Microcomputadoras, Instituto de Ciencias, Universidad Autónoma de Puebla. Recuperado de: Vínculo 16

- Adamatzky, Andrew (2010). Game of Life Cellular Automata. Bristol, United Kingdom. Springer. Recuperado de: Vínculo 17

- Białynicki-Birula, Iwo; Białynicki-Birula, Iwona (2004). Modeling Reality How Computers Mirror Life. New York, US. Oxford University Press.
Recuperado de: Vínculo 18

- López-Borrull, Alexandre; Vives-Gràcia, Josep y Joan-Isidre, Badell (2018). Fake News, ¿Amenaza u Oportunidad para los Profesionales de la Información y la Documentación? Fake news, threat or opportunity for information professionals?.
Recuperado de: Vínculo 19

- Brauer, Fred; Castillo-Chávez, Carlos; De La Pava, Elmer; Barley, Kamal; Castillo-Garsow, Carlos W.; Chowell, Diego; Espinoza, Baltazar; Gonzalez Parra, Paula; Hernandez Suarez, Carlos; Moreno, Victor M. (2015). Modelos De La Propagación De Enfermedades Infecciosas. Universidad Autónoma de Occidente. Cali, Colombia. Recuperado de: Vínculo 20

- Ortega, J; Anguita, M; Prieto, A. (2005). Arquitecturas de Computadores. Madrid, España. International Thomson Editores Spain.

- Suarez Alvarez, Mariano (2011). Autómatas Celulares. Departamento de Matemática Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. Buenos Aires, Argentina. Recuperado de: Vínculo 21

- Yiran, Gu; Jinzhu, Ding (2012). Research on Rumors Spread Based on Cellular Automata. Proceedings of the 2nd International Conference on Green Communications and Networks, editores: Yang, Yuhang; Ma, Maode (GCN 2012): Volume 1. ISBN 978-3-642-35418-2. Londres, Inglaterra. Springer Heidelberg.

- Maitanmi, Olusola Stephen; Adekunle, Yinka y Agbaje, Michael (2013). Model-Based Cellular Automata on Spread of Rumours. Computer Science, Babcock University. Ilisan Remo, Ogun State, Nigeria.
Recuperado de: Vínculo 22

- Flynn Michael J. (1995). Computer Architecture Pipelined and Parallel Processor Design. Londres, Inglaterra. Jones and Bartlett Publishers.
- M. J. Peredo y R. Ramallo (2003). Aplicación de Autómatas Celulares a Simulación Básica de Incendios Forestales, Cochabamba. Recuperado de: Vínculo 23

- Caparrini, F. S. (2016). Autómatas Celulares. Recuperado de: Vinculo 24

- Martínez, G. J. (2006). Introducción a la simulación de procesos con autómata celular. México DF. Recuperado de: Vínculo 25

- Guedez, D. (2005). Autómatas Celulares. Caracas, Venezuela. Recuperado de: Vínculo 26

- Gardner, M. (1970), Mathematical Games. Recuperado de: Vínculo 27

- Kermack, William; McKendrick A. y Walker, Gilbert (1997). A contribution to the mathematical theory of epidemics. Proceedings Royal Sociaty. Vol 115, Issue 772. Pp 700–721. London.

- TED (3 de abril del 2015). Bill Gates: ¿La próxima epidemia? No estamos listos. Recuperado de: Vínculo 28

- Real Academia Española [RAE] (2020). Definición. Edición Tricentenario. *Rae.es*. Recuperado de: Vínculo 29

- D. Kirk y W. Hwu (2010). Programming Massively Parallel Processors, A Hands on Approach. Elsevier, Morgan Kaufmann.

APÉNDICE A

MPI Y Python

El lenguaje de programación Python es uno de los lenguajes de programación que permite la inclusión de la biblioteca MPI para poder escribir programas paralelos, la más interesante y utilizada es la biblioteca *mpi4py*. Es una biblioteca cuya curva de aprendizaje es muy baja si uno es un programador MPI C.. Por lo tanto, su uso es muy amplio en Python (Saccone, 2015).

Las principales aplicaciones del módulo son las relacionadas a la comunicación, provee funciones para dos tipos, ellas son:

- Comunicación punto a punto.
- Comunicación colectiva.

En MPI, los procesos involucrados en la ejecución del programa paralelo tienen un identificador único, el cual se inicia en 0 y finaliza en la cantidad de procesos menos 1. Además todos forman parte del comunicador general MPI.COMM_WORLD. Veamos a continuación el clásico ejemplo del *"¡Hola, mundo!"* donde cada proceso escribe en la consola un mensaje. Como puede observarse, en este código no existen comunicaciones entre procesos, cada uno trabaja en forma totalmente independiente del resto.

```
#hello.py
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
print("Hola mundo desde el proceso ", rank)
```

Para ejecutar un programa en python con MPI, en una terminal se invoca el comando mpiexec con los parámetros adecuados. En la siguiente línea se realiza la ejecución de "¡Hola, mundo!".

```
> mpirun -np 5 python helloWorld_MPI.py
```

La ejecución de *"¡Hola, mundo!"* para 5 procesos dará como resultado en la consola los siguientes mensajes.

```
> Hola mundo desde el proceso 1
Hola mundo desde el proceso 0
Hola mundo desde el proceso 2
Hola mundo desde el proceso 3
Hola mundo desde el proceso 4
```

Por lo expuesto, los identificadores de los procesos, registrados en la variable *rank,* toman valores de 0 a 4. Puede apreciarse que la salida estándar no necesariamente estará ordenada, al existir múltiples procesos al mismo tiempo escribiendo en la pantalla, el sistema operativo elige arbitrariamente el orden, a menos que se especifique el orden de impresión (Saccone, 2015).

Podemos distinguir algunas funciones relevantes, necesarias para la mayoría de los desarrollos en MPI. Ellas son:

- *comm.get_rank ()*
  Esta función devuelve el identificador del proceso que la invocó en el comunicador. Donde *comm* es el comunicador:

  comm = MPI.COMM_WORLD.

  en este caso, el comunicador es MPI.COMM_WORLD, el cual contiene todos los procesos al inicio de la ejecución. MPI permite crear, luego durante el programa, otros comunicadores según se requieran.
- *comm.Get_size()*

Esta función retorna la cantidad de procesos existentes en el comunicador *comm*. Si en el ejemplo se hubiera puesto la línea de código

nproc = comm.Get_size()

la variable *nproc* para el ejemplo de ejecución tendría el valor 5.

Para el caso de las comunicaciones, es posible distinguir las comunicaciones punto a punto, las cuales permiten el envío de un mensaje desde un nodo emisor a uno receptor específicamente Saccone (2015) y Dalsin (2019). Este mecanismo permite la transmisión de datos entre un par de procesos.

El módulo python mpi4py permite la comunicación punto a punto a través de dos operaciones básicas:

- Envío(*dato*, destino): Envía *dato* al proceso *destino.*
- Recepción(dato, origen): Recibe *dato* del proceso de *origen*.

Estas dos operaciones básica se llevan a cabo siempre dentro de un comunicador, en el cual es válido los identificadores de procesos especificados en *destino* y *origen*.

Las comunicaciones punto a punto tienen dos modos de ejecución, ellos pueden ser bloqueante y no bloqueante. Para el primer caso, la función bloquea la ejecución del proceso invocante hasta que pueda reutilizar de forma segura las memorias intermedias (buffer) de datos involucradas en la comunicación. Por el contrario las comunicaciones no bloqueantes no detienen la ejecución del proceso, si los datos están son enviados o recibidos, sino continua con la ejecución del proceso. Estas funciones nos permiten superponer cómputo y comunicación. Generalmente la comunicación sin bloqueo se desarrolla en dos partes en el programa:

- Parte 1: Realiza la función de comunicación propiamente dicha.
- Parte 2: Se verifica si las operaciones realizadas fueron completadas. Esta parte puede existir o no, todo depende de la aplicación a resolver.

En MPI para python, los métodos *MPI.Comm.Send(), MPI.Comm.Recv()* y *MPI.Comm.Sendrecv()* corresponden a comunicaciones bloqueantes; mientras que *MPI.Comm.Isend ()* y *MPI.Comm.Irecv()* son para realizar operaciones no bloqueantes. Estos últimos métodos devuelven una instancia *MPI.Request*, la cual identifica de forma exclusiva la operación iniciada. Su finalización se puede administrar mediante

MPI.Request.Test(), MPI.Request.Wait() y MPI.Request (Dalsin, 2019)(Saccone, 2015)(Almeida et al, 2008)

El siguiente ejemplo muestra el código de un programa *"¡Hello World!"* con comunicaciones punto a punto bloqueante.

```
1.  from mpi4py import MPI

2.  comm = MPI.COMM_WORLD
3.  rank = comm.Get_rank()
4.  size = comm.Get_size()

5.  if rank == 0:
6.    msj = "¡Hola!, el proceso 0 ha enviado un mensaje."
7.    for i in range(rank+1, size):
8.        comm.send(msj, dest=i)
9.  elif rank != 0:
10.   msj = comm.recv(source=0)
11.   print('Mensaje para proceso',rank,':',msj)
```

En las líneas 8 y 10 se puede observar la invocación a cada una de las funciones.
Si se ejecuta el programa para 4 procesos.

```
> mpirun -np 4 python3 ejemplo.py
```

Una salida por consola sería

```
> Mensaje para proceso 3 : ¡Hola!, el proceso 0 ha enviado un mensaje.
Mensaje para proceso 1 : ¡Hola!, el proceso 0 ha enviado un mensaje.
Mensaje para proceso 2 : ¡Hola!, el proceso 0 ha enviado un mensaje.
```

El proceso 0 no imprime porque él envía el mensaje a los *p-1* procesos restantes, y son ellos los que reciben e imprimen el mensaje enviado.

Otro tipo de comunicaciones en MPI son las comunicaciones colectivas. Estas permiten transmitir datos entre múltiples procesos de un comunicador en simultáneamente. La sintaxis y la semántica de las funciones colectivas es similar a las comunicaciones punto a punto. Son comunicaciones bloqueantes y su selectividad está implícita en el orden de llamada (Dalsin, 2019). Entre las comunicaciones colectivas más utilizadas encontramos (ejemplos y gráficos extraídos de (Saccone, 2015)):

- Barrier: Es una sincronización de barreras entre todos los miembros del comunicador. Su sintaxis es

  Comm.Barrier()

  donde Comm es el comunicador. En la Figura 5.1 se ilustra su funcionamiento.

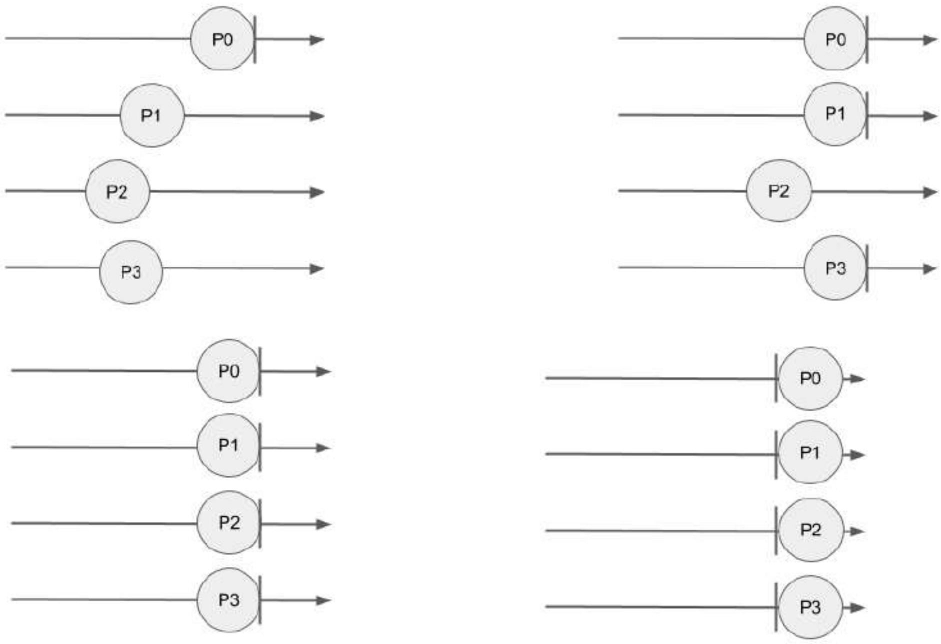


Figura 5.1 Barrera de procesos.
Extraída de: Zaccone, Giancarlo (2015). Python Parallel Programming Cookbook. Packt Publishing. Birmingham, Reino Unido. Página 93.

- Broadcast: Transmite los datos de un miembro, denominado raíz a todos los miembros del comunicador (Ver Figura 5.2). La sintaxis es:

  datosRec=comm.bcast(datos, raiz)

donde *datosRec* es el lugar para dejar los datos recibidos, *datos* los enviados, *raíz* el proceso que envía y *comm* el comunicador. Para el ejemplo de la figura, el proceso 0 es la raíz.

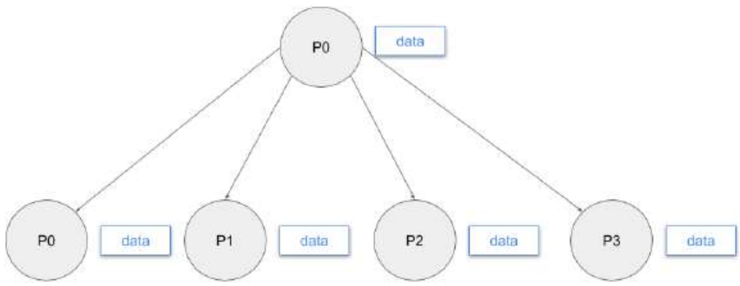


Figura 5.2 Transmitiendo datos del proceso 0 a los procesos 1, 2 y 3.
Extraída de: Zaccone, Giancarlo (2015). Python Parallel Programming Cookbook. Packt Publishing. Birmingham, Reino Unido. Página 108.

El siguiente programa ejemplifica su uso.

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    valor = 100
else:
    valor = None
valor = comm.bcast(valor, root=0)
print("Proceso ", rank," valor:", valor)
```

El comando para su ejecución y la salida para 4 procesos se muestra a continuación.

> mpirun -np 4 python3 bcast.py	> Proceso 0 valor: 100 Proceso 1 valor: 100 Proceso 2 valor: 100 Proceso 3 valor: 100

- Scatter: Distribuye datos desde un proceso raíz a todos los procesos del comunicador (ver Figura 5.3). La sintaxis es:

  datosRec=comm.scatter(datos, raiz)

  donde *datosRec* es el lugar para dejar los datos recibidos, *datos* los enviados, *raíz* el proceso que envía y *comm* el comunicador. Para el ejemplo de la figura, el proceso 0 es la raíz.

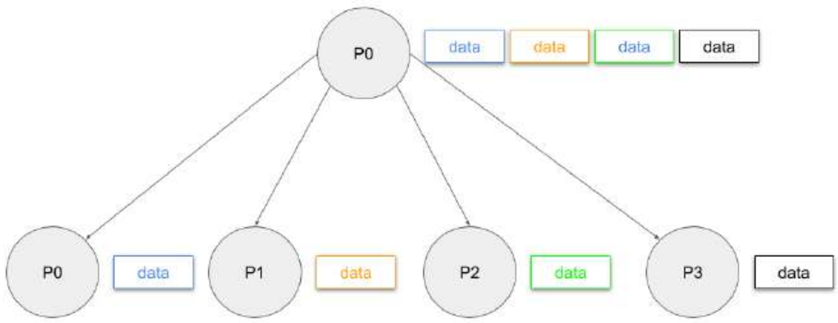


Figura 5.3 Dispersión de datos del proceso 0 a los procesos 1, 2, 3.
Extraída de: Zaccone, Giancarlo (2015). Python Parallel Programming Cookbook. Packt Publishing. Birmingham, Reino Unido. Página 111.

El siguiente programa ejemplifica su uso.

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
if rank == 0:
    array_to_share = [1, 2, 3, 4 ,5 ,6 ,7, 8]
else:
```

```
    array_to_share = None
recvbuf = comm.scatter(array_to_share, root=0)
print("Proceso", rank, "Buffer recibido:", recvbuf)
```

El comando para su ejecución y la salida para 4 procesos se muestra a continuación.

> mpirun -np 8 python3 scatter.py	Proceso 0 Buffer recibido: 1 Proceso 3 Buffer recibido: 4 Proceso 2 Buffer recibido: 3 Proceso 1 Buffer recibido: 2 Proceso 4 Buffer recibido: 5 Proceso 6 Buffer recibido: 7 Proceso 5 Buffer recibido: 6 Proceso 7 Buffer recibido: 8

- Gather: Recopila todos los datos de los miembros del comunicador en el proceso raíz (ver Figura 5.4). La sintaxis es:

  datosRec=comm.gather(datos, raiz)

  donde *datosRec* es el lugar para dejar los datos recibidos, *datos* los enviados por cada proceso, *raíz* el proceso que recolecta y *comm* el comunicador. Para el ejemplo de la figura, el proceso 0 es la raíz.

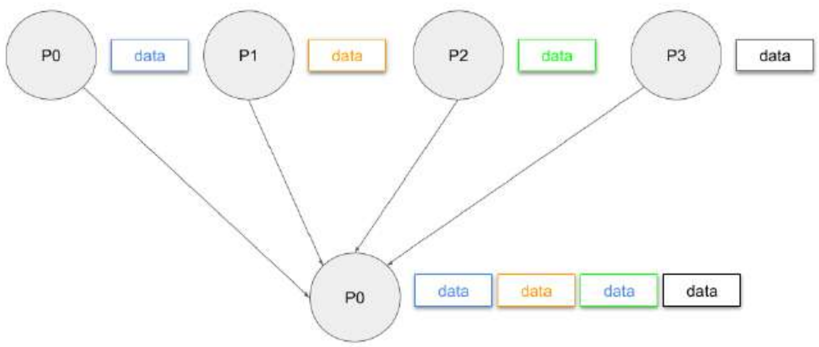


Figura 5.4 Recopilación de datos de los procesos 1, 2, 3.

Extraída de: Zaccone, Giancarlo (2015). Python Parallel Programming Cookbook. Packt Publishing. Birmingham, Reino Unido. Página 114.

El siguiente programa ejemplifica su uso.

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
data = (rank+1)**2

data = comm.gather(data, root=0)

if rank == 0:
    print("rank", rank, "...receiving data to other process")
for i in range(1,size):
    data[i] = (i+1)**2
    value = data[i]
    print("proceso",rank,"recibe", value," del proceso",i)
```

El comando para su ejecución y la salida para 4 procesos se muestra a continuación.

> mpirun -np 5 python3 gather.py	rank 0 ...recibiendo datos a otros procesos proceso 0 recibe 4 del proceso 1 proceso 0 recibe 9 del proceso 2 proceso 0 recibe 16 del proceso 3 proceso 0 recibe 25 del proceso 4

- Reduce: Recopila todos los datos de los miembros del comunicador en el proceso raíz y los reduce según la operación especificada. Esta puede ser suma, máximo, mínimo, etc. Gráficamente se muestra un ejemplo en la Figura 5.5. La sintaxis es:

comm.gather(datos_env, datos_rec, raiz, operación)

donde *datos_env y datos_rec* son los datos enviados y recibidos, *raíz* el proceso que recolecta y opera, *operación* a aplicar sobre los datos y *comm* el comunicador. Para el ejemplo de la figura, el proceso 0 es la raíz.

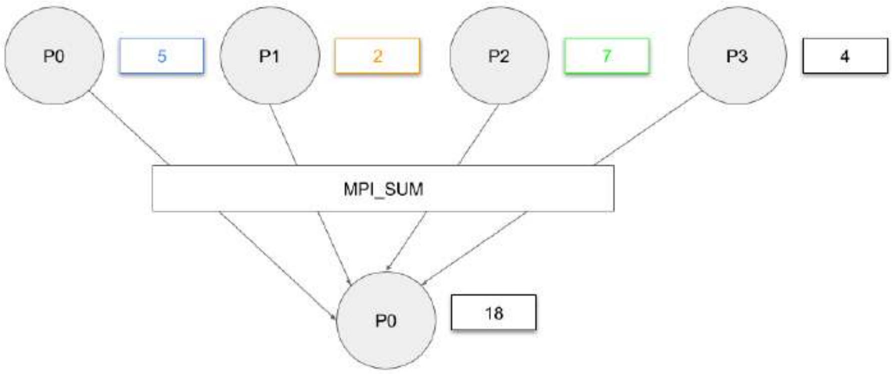


Figura 5.5 La reducción de la comunicación colectiva.
Extraída de: Zaccone, Giancarlo (2015). Python Parallel Programming Cookbook. Packt Publishing. Birmingham, Reino Unido. Página 119.

Supongamos que deseamos realizar la suma de las numeraciones de rank entre 0 y N-1 procesos mediante una operación de reducción (MPI.SUM), utilizando la funcionalidad de Reduce(). Si cada proceso aporte su identificador de rank.

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.rank
sendmsg = rank
recvmsg = comm.reduce(sendmsg, op=MPI.SUM, root=0)

if (rank == 0):
    print(“Total: ”recvmsg)
```

La ejecución y salida correspondiente serían:

> mpirun -np 5 python3 reduce.py	Total: 10

En este Apéndice hemos detallado las principales funciones de MPI para python, para más detalle remitirse a (Saccone, 2015).

APÉNDICE B

Simulación Dispersión Epidemia de Presentada por Bill Gates

En la conferencia TED (3 Abril 2015) en la ciudad de Vancouver (Canadá), el cofundador de Microsoft, Bill Gates, disertó una charla en la cual comenzó con la ayuda de un instrumento, “gran barril negro” con los sellos del Departamento de Defensa de Estados Unidos. El barril, explicó, era lo que muchas familias utilizaban para guardar en su sótano comida enlatada, agua y otros artículos necesarios para sobrevivir la gran amenaza de su época: una guerra nuclear.

Pero, quito el foco de la conferencia del “apocalipsis atómico”, para dar interés lo que, para él, sería el próximo gran riesgo de global catástrofe: una pandemia causada por un virus altamente infeccioso que se propagara rápidamente por todo el mundo y contra el cual no estaríamos listos para luchar.

La charla se dió en contexto de la epidemia de ébola que, entre 2014 y 2016, cobró unas 10.000 vidas y afectando principalmente a África Occidental antes de extenderse ilimitadamente a otros países, como Estados Unidos, Italia y España.

Durante la charla, el disertante expuso una simulación en una pantalla gigante que reproduce el modelo de virus que se propaga por el aire como la gripe Española en 1918 (Figura 3.1).

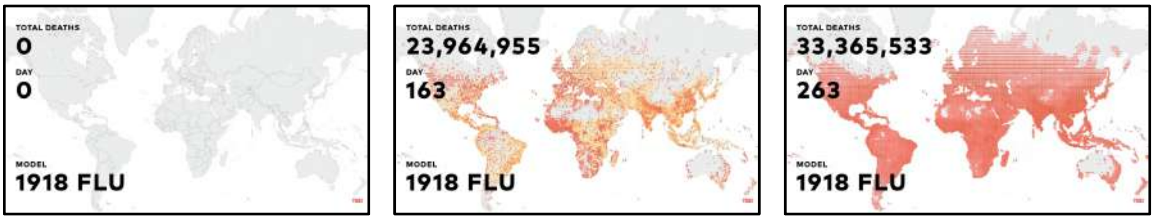


Figura 3.1 Simulación Dispersión Epidemia de Presentada en la Conferencia.
Fuente: Figura 3.1

El espectro de colores varía de una zona aún sana sin caso confirmados de la gripe, zonas grises, intensificando la gravedad de la dispersión del virus, zonas naranjas y rojas (Figura 3.2)

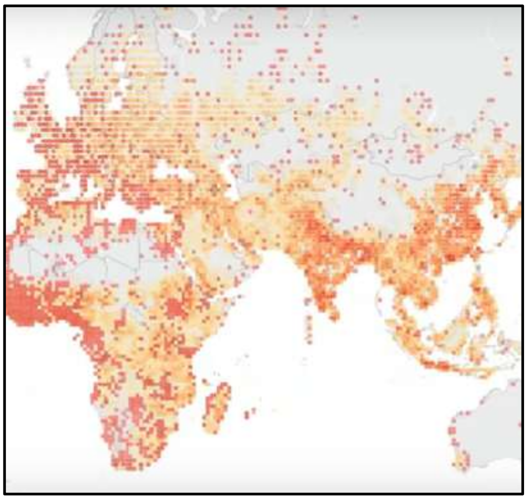


Figura 3 Apreciación de Simulación de Estados Epidémicos
Fuente: Figura 3.2