TEC | Tecnológico de Costa Rica

Diseño e Implementación de un Modelo del Protocolo de Coherencia de Caché MOESI en Sistemas Multiprocesador

Área Académica Ingeniería en Computadores
CE4302 — Arquitectura de Computadores II

Integrantes:
Michael Shakime Richards Sparks

Prof. Luis Barboza Artavia

Cartago, Costa Rica

Abril, 2022

Índice

1. Descripción

A fin de aplicar los conceptos de arquitectura de computadores, se plantea el diseño de un modelo en software del protocolo de coherencia de caché MOESI para un sistema multiprocesador simulado. El MOESI

El protocolo de coherencia de caché es un mecanismo que supervisa la presencia de datos en las cachés de los procesadores centrales, identifica cuándo un procesador requiere datos de otro y cuando varios procesadores comparten información, y optimiza el uso de ancho de banda de memoria

al reducir el tráfico generado. En resumen, el protocolo de coherencia de caché es una solución efectiva para gestionar el flujo de datos entre los procesadores y la memoria, mejorando el rendimiento y la eficiencia del sistema en general.

2. Listado de requerimientos del sistema

En esta sección se detallarán los requerimientos planteados, las herramientas de ingeniería modernas empleadas en el desarrollo de la aplicación, así como estándares, normas y el estado de arte de los recursos utilizados.

- **Sistema multiprocesador**
  - Se requiere un sistema multiprocesador con cuatro procesadores, cada uno debe operar en hilo de forma independiente.
  - Se requiere que cada procesador tenga una memoria caché local L1 con 4 bloques.
  - Se requiere la implementación de un módulo controlador dentro de cada procesador que genere solicitudes a un único bus, dependiendo de la instrucción en ejecución.
  - Se requiere que cada una de las instrucciones sea generada utilizando una distribución de probabilidad formal, sin el uso de bibliotecas.
  - Se requiere la creación de un modulo de bus, que controle las peticiones de los procesadores y les brinde la información solicidata en cada caso.
- **Memoria principal**
  - Se requiere una memoria principal unificada y compartida que se comunique con los procesadores mediante un único bus.
  - Se requiere que la memoria principal actualice el contenido compartido de los bloques en escrituras según una política de write-back.
  - Se debe simular la latencia propia de la memoria en el sistema.
  - Se requiere que el contenido de los bloques de memoria sea de 16 bits en representación hexadecimal e inicializado en cero.

- **Memoria Cache L1**
  - Se requiere una cache con correspondencia asociativa por set two-way.
  - Se requiere que la cache tenga información sobre el número de bloque, el estado de coherencia (M, S, I, E, O), dirección de memoria y dato de 16 bits en hexadecimal.
  - Se requiere realizar desaciertos obligatorios en la caché al iniciar la aplicación, el contenido de los bloques de caché debe empezar en cero.
  - Se requiere incorporar una política de invalidación para las cachés de los demás procesadores ante una actualización en alguna caché o un miss por invalidación.
- **Interfaz de usuario**
  - Se requiere una interfaz gráfica que permita visualizar el comportamiento de cada uno de los procesadores al generar las instrucciones, así como el sistema de coherencia de caché.
  - Se debe mostrar en la interfaz gráfica el identificador único de cada procesador.
  - Se debe mostrar en la interfaz gráfica la información detallada de la caché L1 para cada procesador, incluyendo su estado actual, datos almacenados y etiquetas.
  - Se debe mostrar en la interfaz gráfica la última instrucción generada y la instrucción en ejecución para cada procesador.
  - Se debe mostrar en la interfaz gráfica el contenido de la memoria principal.
  - Se deben mostrar alertas de miss tanto de lectura como de escritura en la interfaz gráfica para que el usuario pueda identificar los eventos de falta de caché.
  - Se debe implementar dos modos de visualización temporal en la interfaz gráfica: *Modo paso a paso*, donde el sistema ejecutará solo el siguiente ciclo y se detendrá. *Modo ejecución continua*, donde el sistema no se detendrá automáticamente hasta que el usuario lo detenga manualmente con una opción de pausa.
  - Se debe permitir al usuario agregar una instrucción en modo pausa para el siguiente ciclo, indicando procesador, tipo de instrucción, direcciones y datos.

3. Elaboración de opciones de solución al problema

Con respecto al problema planteado, se han discutidos varias propuestas de diseño, esto tomando en consideración aspectos como la necesidad de utilizar hilos, los requerimientos, el flujo de trabajo, complejidad de implementación y la estructura de los modelos.

3.1. Propuesta 1:

Se plantea utilizar Python como lenguaje de programación debido a sus ventajas. Python es conocido por su flexibilidad y versatilidad, lo que permitirá adaptar fácilmente el código a los requisitos específicos del sistema multiprocesador. Por otra parte, la amplia disponibilidad de bibliotecas, simplifica la implementación de protocolos de comunicación y simulación pues el módulo *threading* de Python proporciona una interfaz sencilla y fácil de usar para trabajar con hilos. Además, se considera el uso del paradigma de objetos, al ser un sistema multiprocesador con varios núcleos, cada uno con su respectiva caché y controlador, este paradigma favorece al proveer modularidad, reusabilidad y lo más importante la abstracción.

Respecto a la implementación del modelo de software se plantea utilizar el patrón de diseño *Observer*, para mantener a los controladores actualizados sobre los cambios en el estado de los bloques de memoria compartida, y permitir una comunicación eficiente entre los nodos y los controladores.

Para este sistema en especifico, el Bus del sistema actuará como el controlador de eventos (figura 1), encargado de gestionar la comunicación entre los nodos del sistema multiprocesador. Cada uno de los nodos tendrá su propio Controlador, que actuará como un observador. Los Controladores de los nodos implementarán una interfaz Observador, que tendrá métodos de actualización para recibir notificaciones sobre los cambios en los bloques de memoria compartida, así como para notificar al Bus cuando ocurra un miss en su caché local o alguna invalidación que deba realizarse.

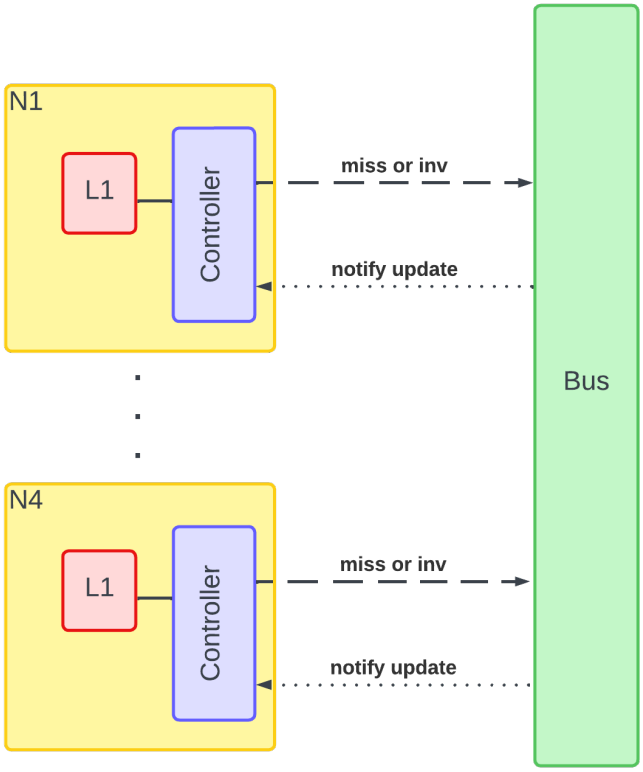

Figura 1: Diagrama de la propuesta 1. Observer

Para esta aproximación, el Bus deberá mantener una lista de nodos para poder notificarles los cambios en los bloques de memoria y una cola para recibir las notificaciones de misses e invalidaciones. De tal forma que los Controladores de los nodos implementarían un método adicional en su lógica interna para cargar un mensaje a la cola con los datos relevantes. Y el bus por su parte, cuando el estado de un bloque de memoria compartida cambie debido a una petición de escritura o a una invalidación, notificará a los nodos observadores en la lista, utilizando los métodos de actualización a implementar. Al recibir las notificaciones del Bus, los Controladores de los nodos actualizarán su propio estado interno y tomarán las acciones necesarias según el protocolo MOESI. Por ejemplo, podrían invalidar su caché local si el bloque de memoria ha sido modificado en otro nodo.

3.2. Propuesta 2:

En esta propuesta de igual manera se desea emplear Python como lenguaje de programación, el mismo paradigma, con la diferencia de que en esta aproximación se planea utilizar una cola de eventos en

el Bus del sistema, donde los Controladores de los nodos encolan eventos de acciones y el Bus actúa como un administrador de eventos para procesar y coordinar las acciones necesarias en el sistema (figura 2).

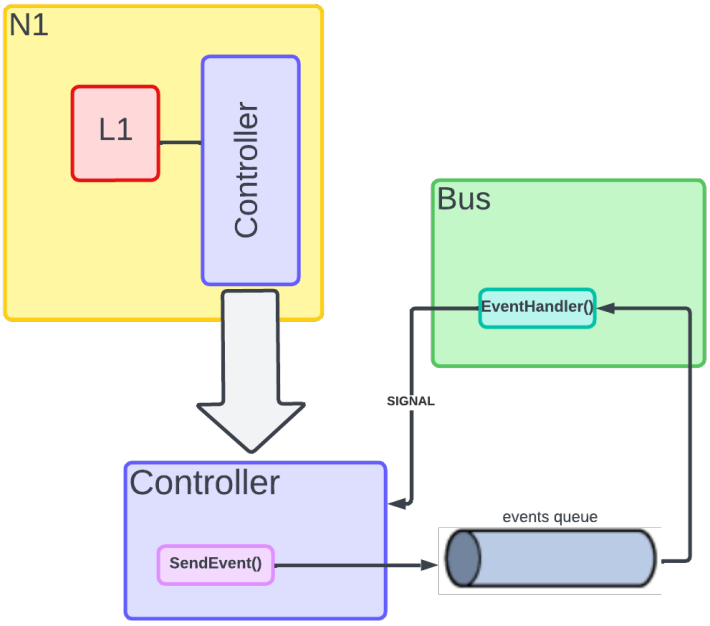

Figura 2: Diagrama de la propuesta 2. Events

El Bus actúa como un intermediario centralizado que maneja los eventos de miss e invalidación. Cuando un Controlador identifica un miss en su caché local o hace una invalidación, en lugar de notificar directamente al Bus o a los demás Controladores, genera un evento y lo coloca en la cola de eventos del Bus. Cuando el Bus procesa un evento en la cola de eventos, realiza las acciones necesarias para traer el bloque de memoria faltante a la caché local del Controlador correspondiente y una vez que el bloque de memoria está disponible en la caché local del nodo, el Bus notifica al Controlador mediante una señal que el bloque está disponible. Entonces, el Controlador puede acceder al bloque de memoria en su caché local y actualizarla según sea necesario, para mantener la coherencia.

Esta propuesta permitiría una comunicación asíncrona entre los nodos y el Bus, ya que los Controladores solo encolarían eventos en lugar de llamar a métodos directamente. Además, proporcionaría una mayor flexibilidad en el manejo de eventos, ya que el Bus podría gestionarlos de manera más eficiente y coordinada en función de su propia lógica de procesamiento.

4. Valoración de opciones de solución

En relación con la salud y seguridad pública, ambas soluciones tienen un impacto insignificante. En cuanto al costo total de vida, la solución con eventos puede tener ventajas en términos de simplicidad de la lógica de notificación y procesamiento de eventos, lo que puede resultar en un menor costo de desarrollo y mantenimiento a largo plazo. Sin embargo, esto dependerá del diseño e implementación específicos. En términos de carbono neto cero, ambas soluciones tienen un impacto mínimo ya que no hay consideraciones específicas de huella de carbono asociadas con el uso de cualquiera de los sistemas.

En términos de recursos, la resolución del evento puede requerir un acoplamiento más estrecho entre el bus y el controlador de nodo, lo que puede comprometer la flexibilidad en términos de lógica de actualización de caché asignada a cada controlador y por consiguiente afectar los recursos necesarios para desarrollar, mantener y actualizar el sistema. Asimismo, ninguna de las soluciones tiene consecuencias culturales, sociales y ambientales directas, ya que son consideraciones técnicas en el diseño e implementación de sistemas multiprocesador.

5. Selección de la propuesta final

La solución del patrón Observer ofrece ventajas significativas en términos de flexibilidad, separación de responsabilidades y potencial de reutilización del código. En primer lugar, proporciona una clara separación de responsabilidades entre el Bus y los Controladores de nodo, lo que evita un acoplamiento excesivo y facilita la comprensión y modificación del código en el futuro. Además, este patrón es ampliamente conocido y utilizado, lo que significa que hay una gran cantidad de recursos disponibles para aprender, implementar y mantener el patrón, lo que puede contribuir a una mayor reutilización y mantenibilidad del código a lo largo del tiempo.

Por ultimo, permite una lógica de actualización de caché personalizada para cada Controlador de nodo, ya que solo se notifica a los observadores registrados, lo que permite una mayor flexibilidad en la forma en que los Controladores manejan los eventos de miss de caché. En este caso no es una gran ventaja puesto que se necesita informar a todos los procesadores pero esto puede ser especialmente útil en sistemas multiprocesador con diferentes políticas de caché en cada nodo.

En contraste, la solución con eventos puede tener algunas desventajas potenciales. En primer lugar, la implementación de un sistema de eventos robusto y eficiente puede ser más compleja en

términos de lógica de notificación y procesamiento de eventos. Esto puede requerir más tiempo y recursos para desarrollar y mantener en comparación con la solución del patrón Observer, por tanto con el tiempo disponible para la realización del proyecto esto puede ser un riesgo.

Asimismo, al tener un mayor acoplamiento entre el Bus y los Controladores de nodo, cualquier cambio en la lógica puede tener implicaciones en múltiples componentes del sistema y encontrar el error puede ser complicado, además de ser menos mantenible. Y por si fuera poco, la falta de estandarización en sistemas o arquitecturas de este tipo por el conocimiento generalizado puede complicar el desarrollo, pues hay distintas alternativas para hacer la implementación.

Según estas consideraciones, la propuesta empleando el bien conocido patrón de observador es preferible en términos de flexibilidad, mantenibilidad, modularización, separación de tareas y potencial para la reutilización del código.

6. Diseño de la alternativa seleccionada

6.1. Impacto del diseño final

Considerando la salud y seguridad pública, el diseño final del sistema multiprocesador con el patrón Observer se ha asegurado de implementar políticas de caché eficientes y coherentes, garantizando que los datos en la memoria compartida estén actualizados y sean consistentes en todos los nodos. Además, se ha realizado una rigurosa revisión de seguridad para prevenir posibles vulnerabilidades y ataques en el sistema, asegurando que solo los Controladores de nodo autorizados tengan acceso a los datos en la caché.

En términos de costo total de vida, el diseño final ha tenido en cuenta el equilibrio entre el rendimiento del sistema y el costo de implementación y mantenimiento. Se han utilizado recursos disponibles y ampliamente utilizados, como el patrón Observer, lo que contribuye a la reutilización de código y a la mantenibilidad del sistema a largo plazo. Además, se ha considerado el consumo de energía del sistema y se ha optimizado para lograr un carbono neto cero, utilizando técnicas de administración de energía eficientes y sostenibles en los nodos del sistema.

6.2. Descripción del Protocolo Diseñado

El protocolo diseñado para el sistema multiprocesador con el patrón Observer utiliza el protocolo MOESI para mantener la coherencia de la memoria compartida en los nodos del sistema. Cada nodo

tiene su propia caché local y se comunica con el Bus para realizar operaciones de lectura y escritura en la memoria compartida.

El uso del patrón Observer en esta implementación permitirá una comunicación eficiente entre los nodos, el Bus y los Controladores, ya que los Controladores solo serán notificados cuando ocurran cambios relevantes en los bloques de memoria compartida o misses en su caché local. Esto puede mejorar la coordinación y el rendimiento del sistema multiprocesador en general.

6.3. Diagrama de Bloques del Modelo del Sistema Multiprocesador

En la figura 3 se observa el diagrama de bloques que muestra la estructura y conexión de los componentes principales del sistema multiprocesador, incluyendo los nodos con sus respectivas cachés locales, el bus de comunicación, y la memoria compartida.

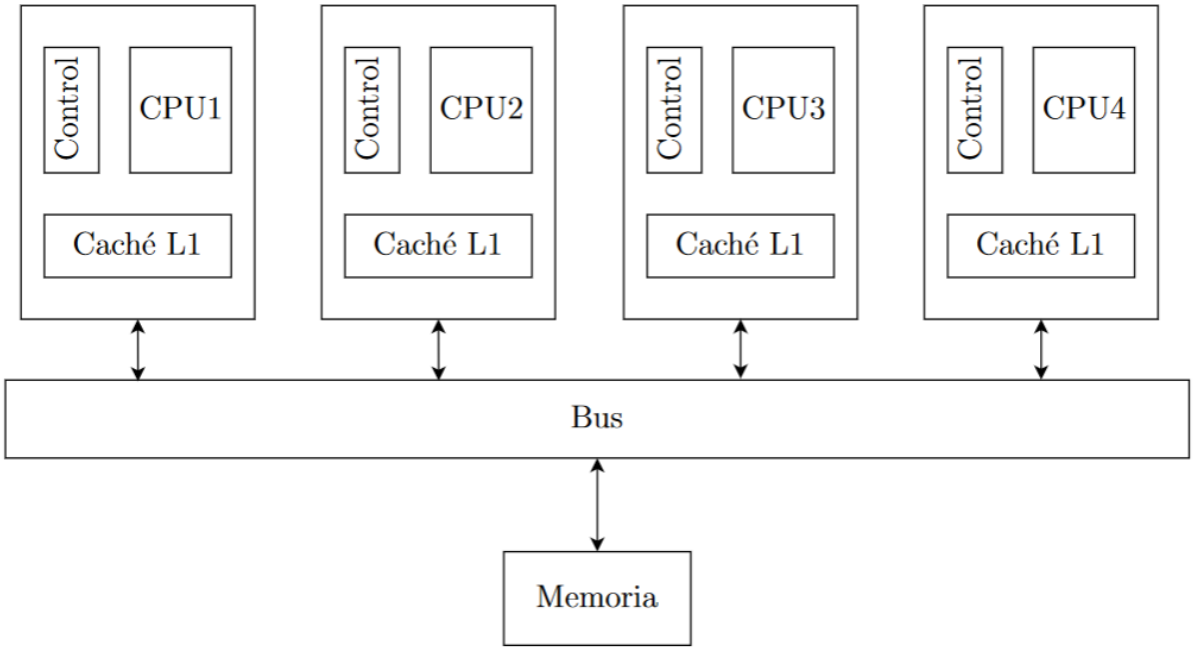

Figura 3: Diagrama de bloques del modelo del sistema multiprocesador

6.4. Diagrama de Bloques del Computador

En la figura 4 se observa el diagrama de bloques del computador el cual muestra cómo se organizan internamente los componentes del procesador, así como cómo interactúa con la memoria caché y cómo

interactúan con la aplicación. Primeramente, el controlador es quien indica a la interfaz cuándo debe realizar alguna animación y es el único quien tiene acceso directo a la memoria caché. Para acceder a la memoria caché es necesario pasar por el controlador del procesador, sucede lo mismo con la memoria principal, no existe un acceso directo, el medio es el bus, quien es el único acceso y quien la protege (ver 3).

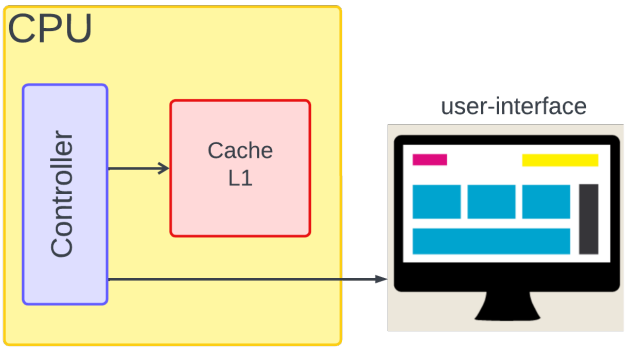

Figura 4: Diagrama de bloques del computador

6.5. Descripción del Algoritmo Propuesto

Para lograr el comportamiento del sistema completo se definieron varios módulos con funcionalidades específica. Se utiliza una combinación de técnicas de administración de caché, control de concurrencia y notificación basada en eventos para mantener la coherencia de la memoria compartida. En la figura 5 se observan los módulos definidos:

- **Random:** Esta clase es un generador de instrucciones aleatorias que utiliza métodos de conversión de números decimales a binarios y hexadecimales, así como generación de números aleatorios. Proporciona funcionalidades como la generación de direcciones de memoria, datos aleatorios y operaciones de lectura, escritura y cálculo. También tiene métodos para establecer los valores de las instrucciones generadas en variables StringVar de la interfaz.

- **CPU:** Esta clase representa una unidad de procesamiento central de la CPU. Tiene un identificador específico. Tiene métodos para ejecutar una instrucción y establecer una nueva instrucción a través de la clase Random. Esta clase utiliza una instancia de la clase Controller para manejar la ejecución de las instrucciones en la memoria caché.

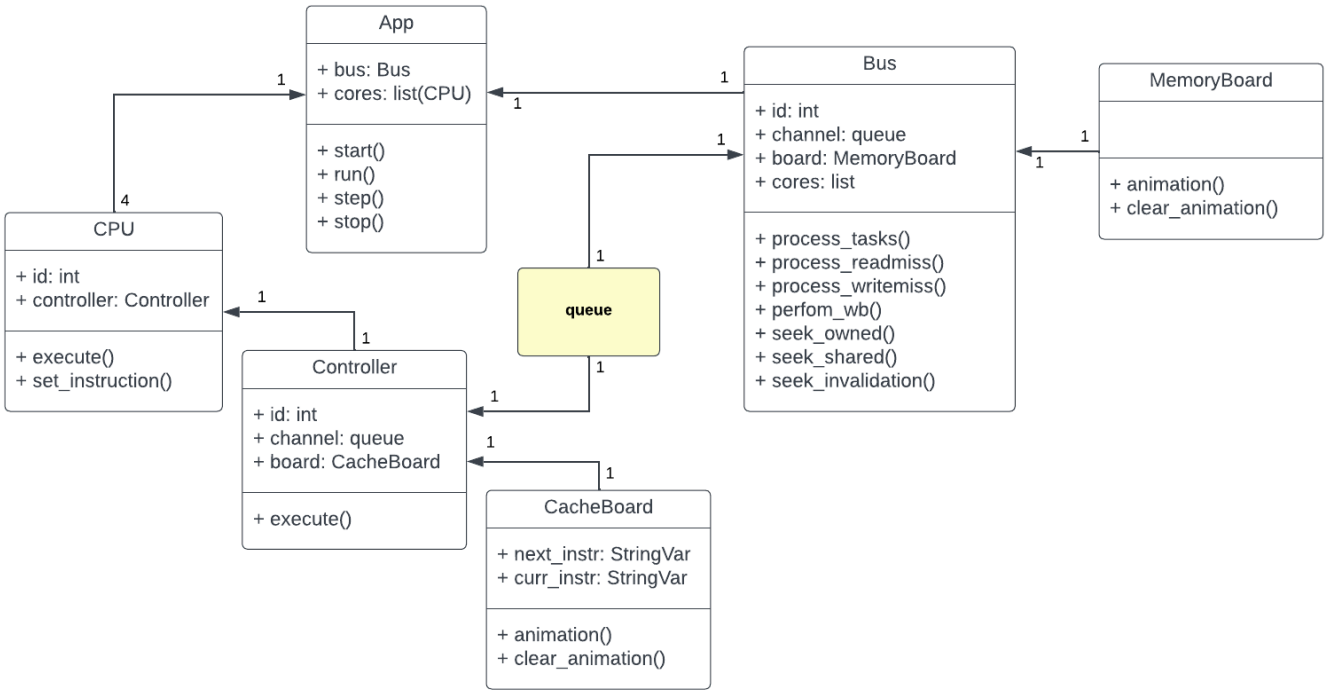

Figura 5: Diagrama de clases

- **Controller:** Esta clase representa el controlador de memoria caché de la CPU. Tiene internamente la referencia a la tabla de interfaz que representa la memoria caché gráficamente y es quien hace cambios sobre esta. Además tiene comunicación con la cola del bus para poder realizar peticiones al canal de comunicación con otros nodos del sistema. Por otra parte, es quien procesa las instrucción a ejecutar, determina la operación a realizar, y se realiza la acción correspondiente en la caché, actualizando su estado y enviando mensajes a través del canal de comunicación.

- **Bus:** Esta clase simula el comportamiento del bus de sistema en un sistema multiprocesador. Se encarga de manejar el procesamiento de mensajes relacionados con la lectura y escritura de bloques de caché en los núcleos de procesador y en la memoria principal. Los mensajes son transmitidos a través de un canal y el bus se encarga de coordinar las operaciones de lectura y escritura entre los diferentes núcleos de procesador y la memoria principal. Implementa la lógica de procesamiento de mensajes, acceso a memoria y además tiene una referencia a la tabla de interfaz que representa la memoria principal, por ende es quien realiza los cambios sobre ella.

- **App:** Esta clase implementa la interfaz gráfica de usuario, la cual permite al usuario interactuar con el programa simulando la ejecución de instrucciones en múltiples CPU, mostrando el estado de ejecución de cada CPU, el estado de la memoria caché y la memoria principal, y permitiendo al usuario controlar la ejecución del programa mediante botones y casillas de verificación.

6.6. Distribuciones de Probabilidad Implementadas

El algoritmo desarrollado para selección de instrucciones utiliza la distribución geométrica para generar las instrucciones y posiciones de memoria de manera aleatoria. La elección de la distribución geométrica se debió a sus características únicas que resultaron ser adecuadas para el propósito específico del algoritmo.

La razón principal detrás de la elección de la distribución geométrica fue la necesidad de generar instrucciones y posiciones de memoria de manera aleatoria, pero con una probabilidad decreciente a medida que aumenta el número de intentos (ver figura 6). Según [1], esta distribución modela la probabilidad de éxito de obtener un elemento específico en cada intento, donde en cada ensayo cada elemento presenta una probabilidad constante, lo cual se cumple en este caso.

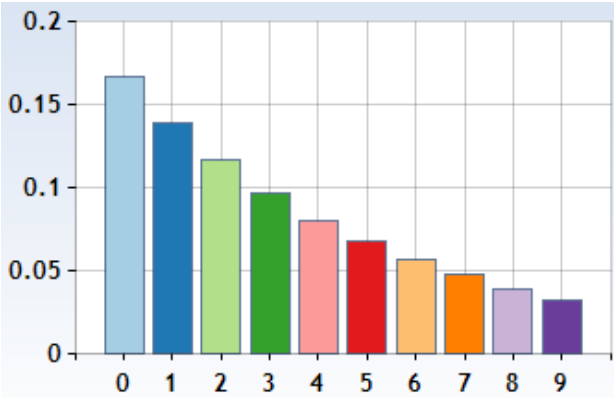

Figura 6: Gráfico de la distribución geométrica.

El algoritmo emplea la distribución para obtener una lista de las probabilidades de cada instrucción y cada dirección, para luego seleccionar el valor menos probable de todos. En otras palabras, se selecciona el elemento menos frecuente a obtener de todos los ensayos realizados, aun cuando en todos los ensayos la probabilidad de salir es la misma para cada elemento.

7. Validación del diseño

7.1. Interfaz Gráfica

La interfaz gráfica del sistema multiprocesador ha sido implementada con éxito, siguiendo principios de usabilidad, como una representación visual clara de los bloques de memoria caché, interacciones intuitivas, feedback en tiempo real, opciones de navegación y visualización optimizadas (ver figura 7. En general, la interfaz ofrece una experiencia de usuario eficiente e intuitiva, mejorando la usabilidad del sistema multiprocesador.

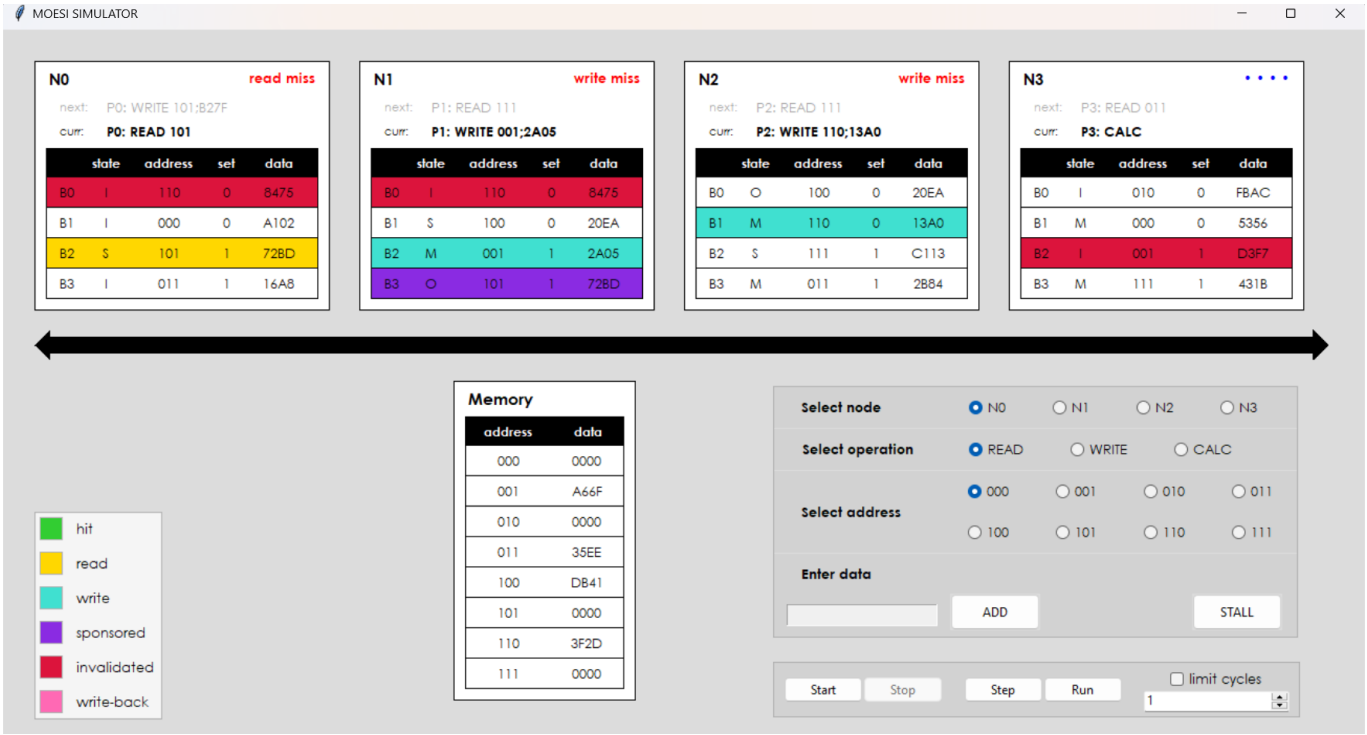

Figura 7: Interfaz de usuario final del modelo propuesto.

Es importante destacar que una interfaz como esta tiene un impacto positivo en la salud de los usuarios. Al ser amigable reduce la carga cognitiva y la frustración del usuario al interactuar con el sistema, lo cual puede mejorar su experiencia y tener un efecto beneficioso en su salud mental. Sin embargo, no solo se limita a mejorar la eficiencia del sistema, sino que también tiene ventajas culturales, como facilitar la educación acerca del protocolo MOESI. La simplicidad de uso puede repercutir positivamente en la adopción y la comprensión activa por parte del usuario.

7.2. Modelo de software

La implementación del modelo de protocolo MOESI se llevó a cabo de manera exitosa, siguiendo un enfoque riguroso en el diseño y desarrollo del sistema de memoria caché distribuida. Se definieron y se implementaron las reglas de transición de estados del protocolo MOESI, incluyendo las operaciones de lectura, escritura, invalidez y actualización de las líneas de caché. Se aseguró la correcta coordinación y sincronización de las acciones entre los distintos nodos del sistema distribuido, manteniendo la coherencia de la información almacenada en las cachés y en la memoria principal.

La implementación exitosa de la integración del sistema gráfico con el protocolo MOESI permitió observar una reducción significativa en los conflictos de acceso a la memoria y en los casos de escrituras redundantes, mejorando así la eficiencia del sistema y evitando problemas de incoherencia de datos. Esto demostró un rendimiento eficiente y una gestión adecuada de los estados de las líneas de caché, lo que contribuyó a una gestión eficiente de la memoria compartida en sistemas multiprocesador.

Como resultado, se comprobó que un protocolo MOESI bien diseñado facilita la colaboración y el intercambio de información en entornos de trabajo colaborativo, lo que puede tener beneficios en términos de productividad y eficiencia. Además, esta gestión eficiente de la memoria compartida promueve la innovación y la creatividad en el uso de recursos culturales y sociales del sistema, lo cual puede contribuir a un mayor desarrollo y provecho de los recursos del sistema multiprocesador.

Adicionalmente, esto tiene un impacto positivo en el ambiente, al optimizar el uso de recursos se minimiza la duplicación innecesaria de datos, evitando conflictos en el acceso a la memoria compartida. Esto puede resultar en una reducción del consumo de energía y recursos del sistema, lo cual puede contribuir a la eficiencia energética y a la reducción de la huella de carbono del sistema multiprocesador.

7.3. Conclusión

En conclusión, la combinación de un diseño eficiente del protocolo MOESI y una interfaz gráfica intuitiva en el sistema multiprocesador puede tener beneficios significativos en términos de experiencia del usuario. Esto no solo mejora la usabilidad del sistema, sino que también puede tener beneficios en la salud mental de los usuarios y promover la educación. Además de la gestión de recursos y sostenibilidad, contribuyendo a un mejor rendimiento y impacto positivo en diversos aspectos del sistema.

Referencias

[1] Taboga, M. (2021). Geometric distribution. https://www.statlect.com/probability-distributions/geometric-distribution.