TEC | Tecnológico de Costa Rica

Propuestas Proyecto I

CE-4302 Arquitectura de Computadores II

Área Académica de Ingeniería en Computadores

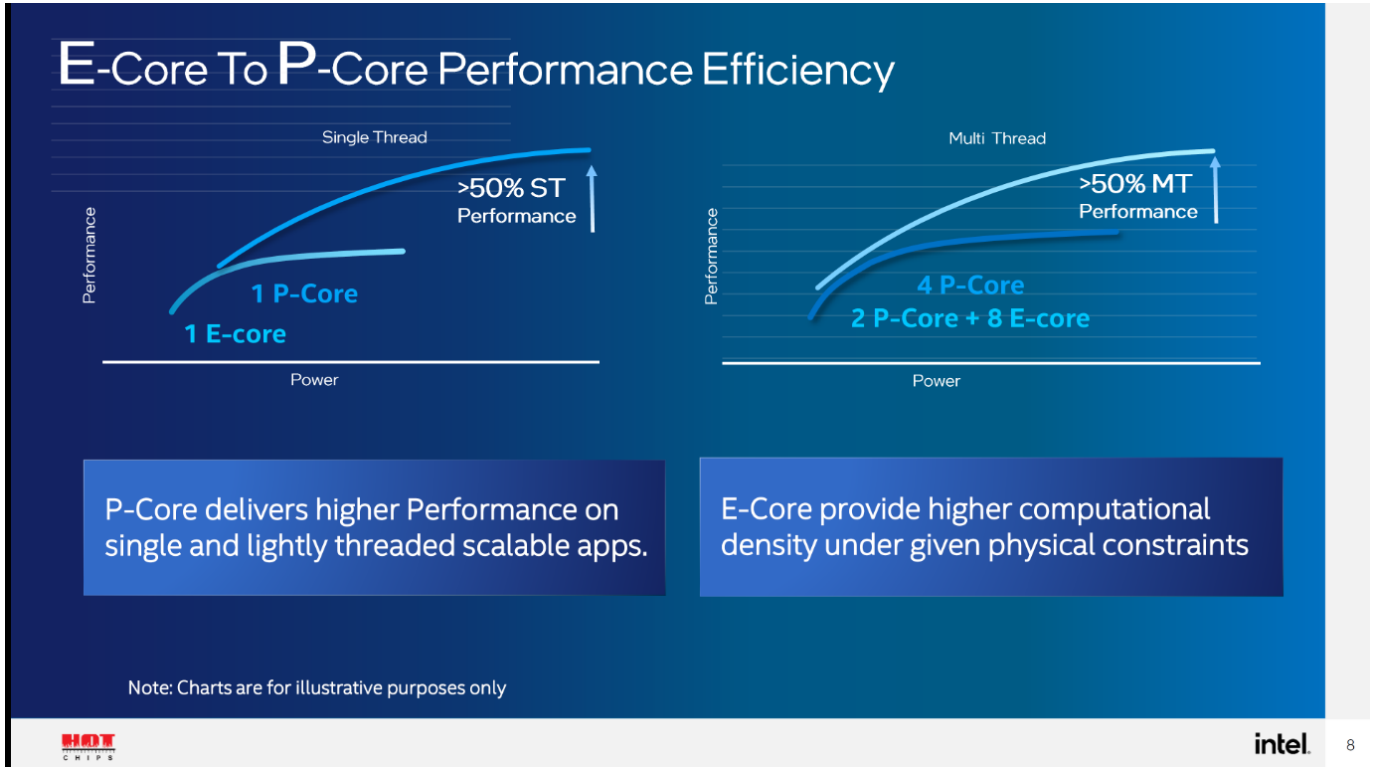
Profesor Ronald García Fernández

Contenidos

- **Motivación**
- **Propuesta 1 (*HW Implementation MESI*)**
- **Propuesta 2 (SW $ Coherence *modelling and evaluation*)**
- **Propuesta 3 (*MP Programming and false sharing*)**

Motivación

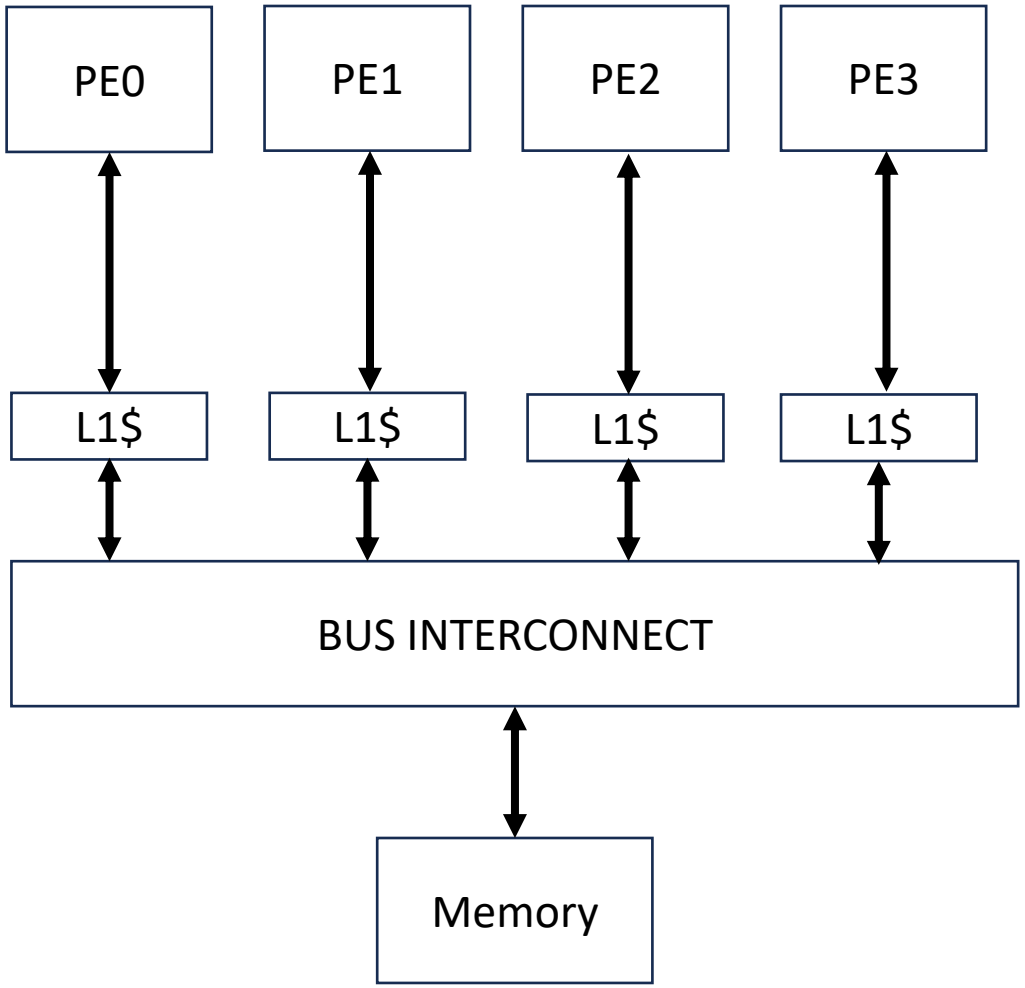
- *TLP* y *MP* permiten mejorar el desempeño de un sistema computacional*
- En la actualidad los sistemas *MP* son la base computación moderna
- *MP* presenta retos a nivel HW y SW



https://wccftech.com/intel-alder-lake-p-core-e-core-detailed-golden-cove-offers-50-higher-single-threaded-hybrid-design-offers-50-higher-multi-threaded-performance/

Propuesta 1 (*HW Modelling MESI Protocol)*

- *Caches* son fundamentales para mejorar el desempeño de un *PE*
- *Cache Coherence* permite el manejo correcto de memoria compartida
- Protocolo MESI es una implementación



Propuesta 1 (*HW Implementation MESI Protocol)*

Objetivos

1- Crear un simple *PE* que escriba y lea datos de memoria (el programa es cargado de .hex) 3 instrucciones: `write addr reg`, `read addr reg, incr reg`

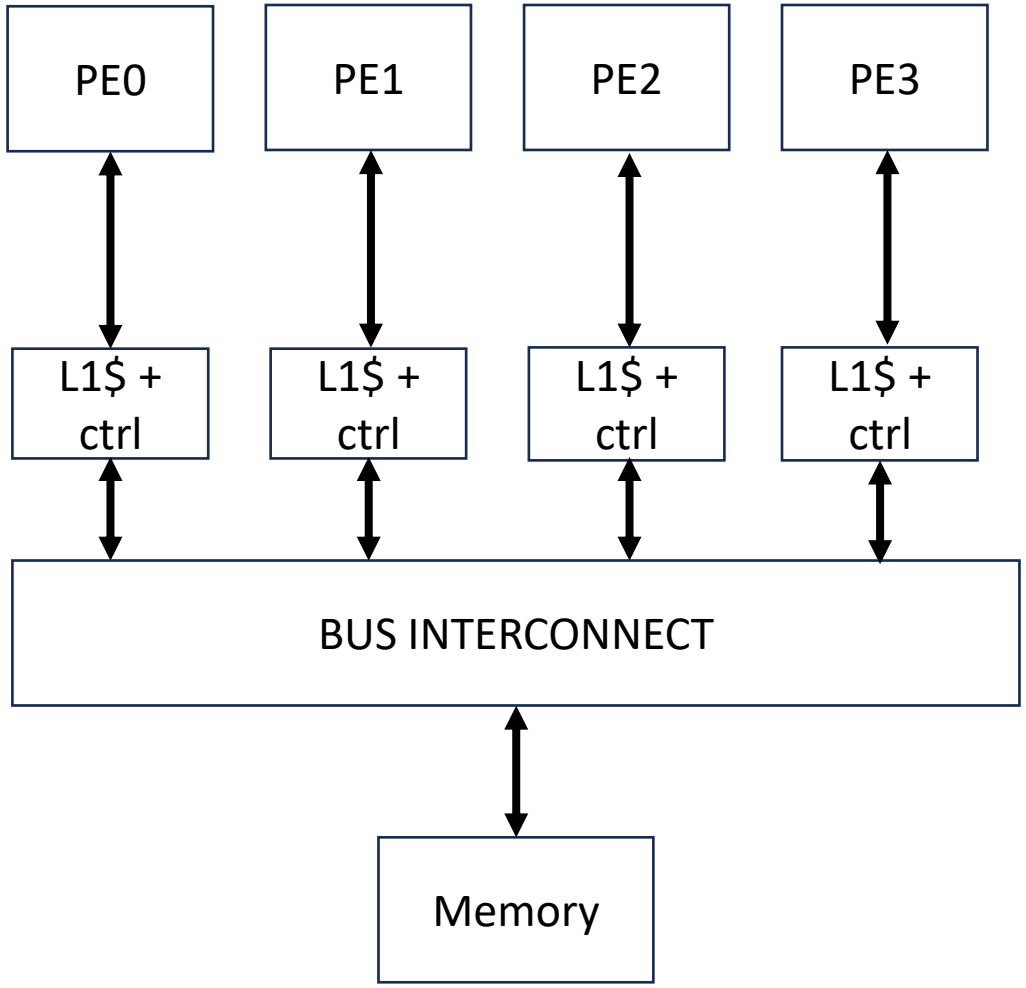
2- Crear un *$* + controlador simplificado con los bits de protocolo *MESI* (V, D, etc) en cada línea de $

3- Implementar el *BUS Interconnect* capaz de enviar mensajes a los otros *PEs* $

4- Implementar una memoria simple que maneje operaciones de lectura y escritura

5- Simular el sistema y crear un conjunto de pruebas para validar los resultados

6- Implementar el sistema en un *FPGA* con interfaz para controlar ejecución de instrucciones (*stepping & start*)



Propuesta 1 (*HW Implementation MESI Protocol)*

Características

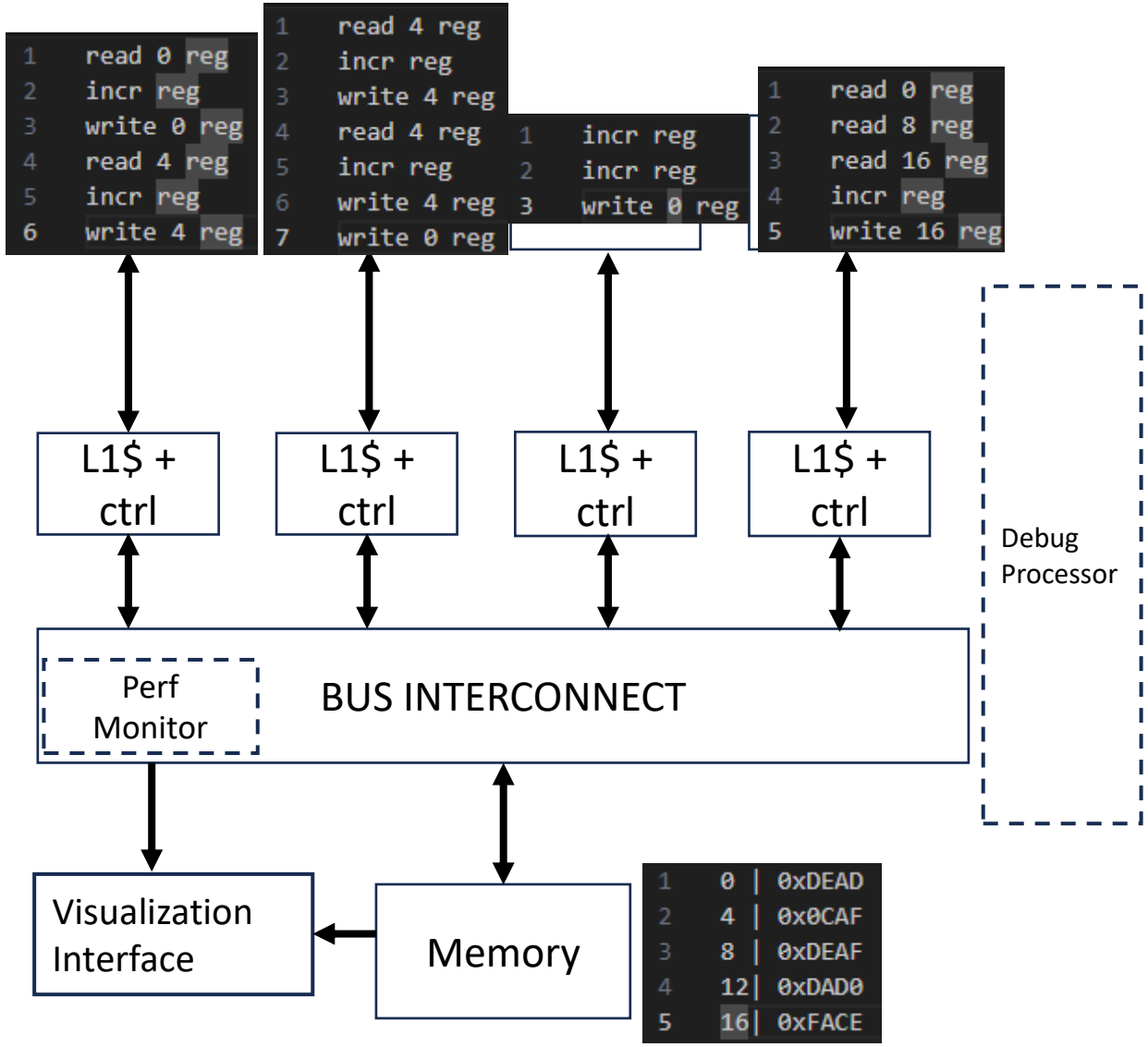
1- El sistema tiene que poseer al menos 3* *PEs*
2- Memoria 16 bits datos 16 *entries* mínimo
3- *$* 4 *entries* de 16 bits
4- $ implementa ***write-back*** *policy*
5- *Interconnect* tiene que serializar operaciones a memoria
6- *Interconnect* tiene manejar envío de mensajes entre *$ controllers*
7- Cada PE debe permitir stepping/start para procesar instrucciones
8- FPGA debe permitir observar los valores en memoria

Extra

9- Lógica de conteo/monitoreo de transacciones (*bus cycles*) por tipo (INV, READ_REQ/RESP, WRITE_REQ/RESP)

10- Lógica de *debug* de estado de línea de cache
Ejemplo

```
PE0 -> L1[0..3] => V=1, D=0, TAG=0x01, data=0xCAFE
```



Propuesta 1 (*HW Implementation MESI Protocol)*

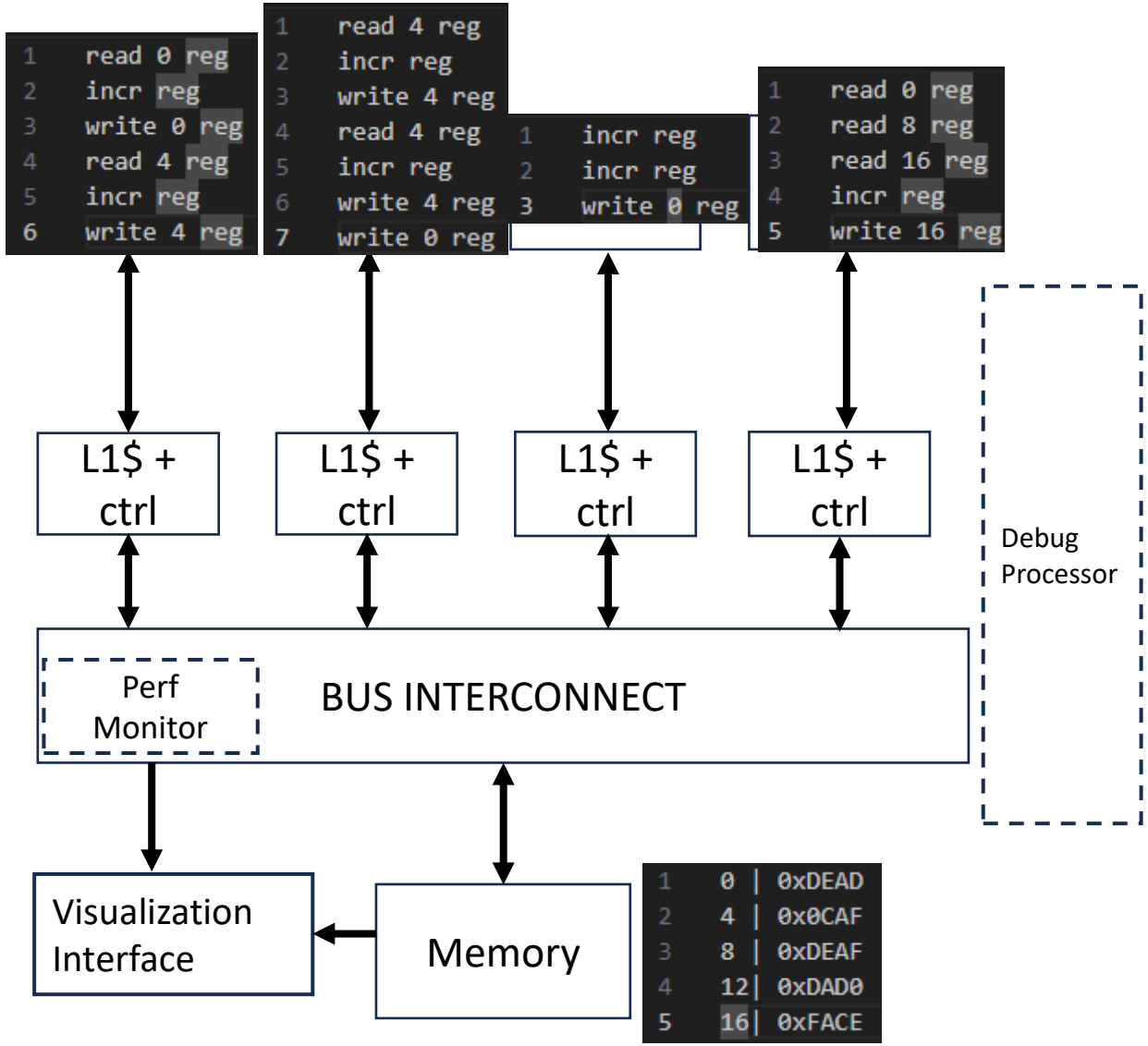
Entregables

1- Propuesta de diseño (evaluación ventajas/desventajas)
2- bitstream del sistema
3- Especificación de componentes del sistema (tipo de comunicación, funcionalidad, limitaciones)
4- Explicación proceso de diseño
5- Demostración funcional
6- Simulaciones de bloque y sistema*
7- Reporte de resultados

Extra:

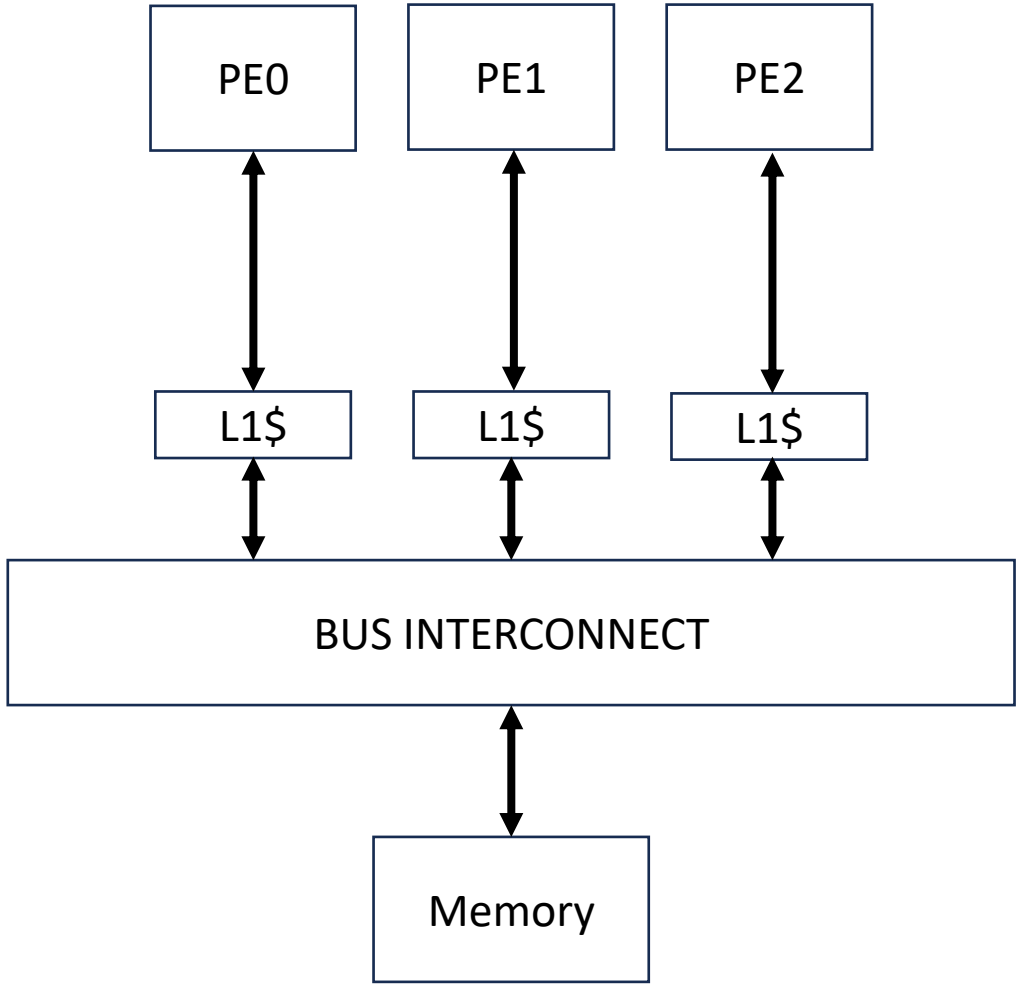
8- Modelo de referencia del sistema para verificación
9- Plan de pruebas y cobertura de funcionalidades*

*Si su grupo es 4 personas el punto 9 es obligatorio



Propuesta 2 (SW $ Coherence *modelling and evaluation)*

- *Caches* son fundamentales para mejorar el desempeño de un *PE*
- *Cache Coherence* permite el manejo correcto de memoria compartida
- Modelar y evaluar diferentes protocolos ejemplo MESI, MOESI y su impacto en el desempeño



Propuesta 2 (SW $ Coherence *modelling and evaluation)*

Objetivos

1- Modelar un sistema *MP* con al menos 3 *PEs* con $ privado y única memoria compartida e *interconnect* empleando *threads*

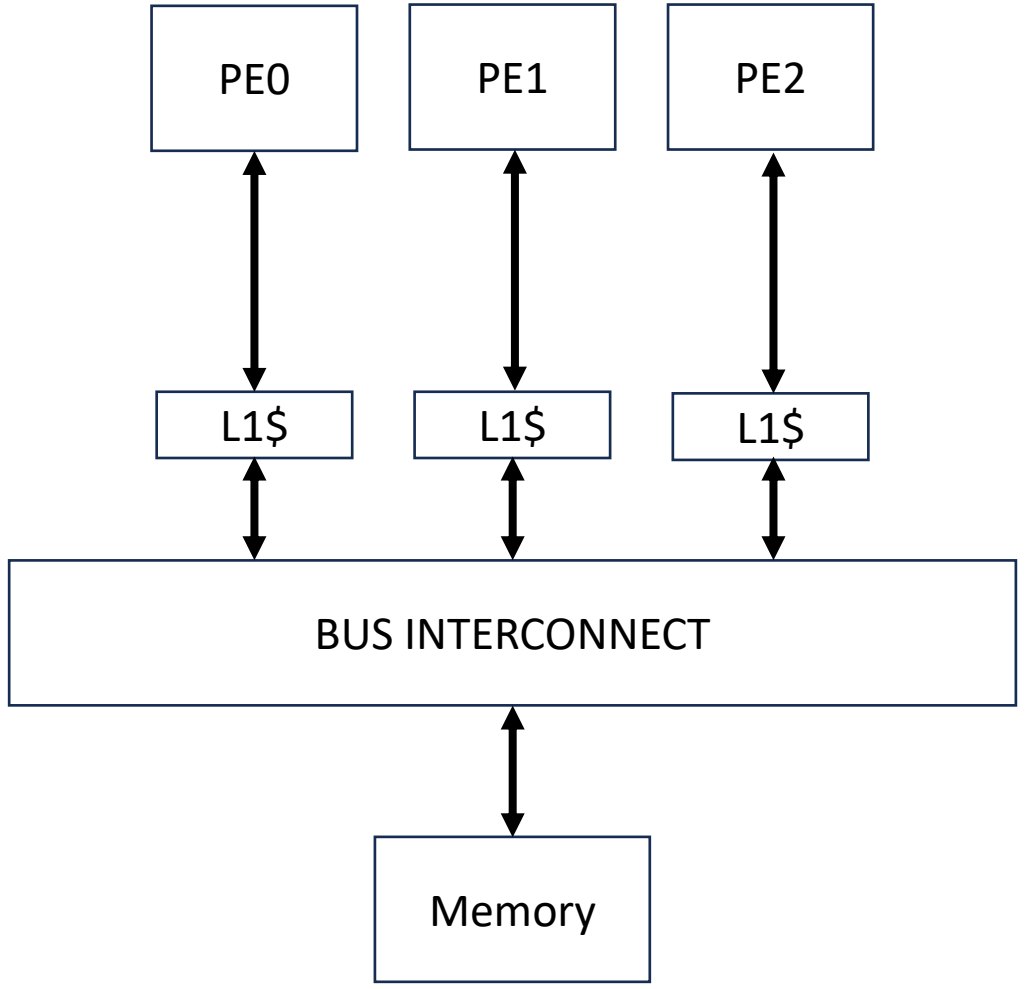
2- Implementar una interfaz gráfica donde se pueda visualizar los estados de las líneas de cache respecto al protocolo a implementar y las transacciones realizadas.

3- Implementar al menos 2 protocolos de coherencia (MESI y MOESI mínimo)

4- Generar aleatoriamente código de prueba (código que cada PE ejecuta)

5- Realizar análisis comparativo sobre resultados de pruebas en el sistema bajo cada protocolo

6- Implementar el sistema con interfaz para controlar ejecución de instrucciones (*stepping & start*)

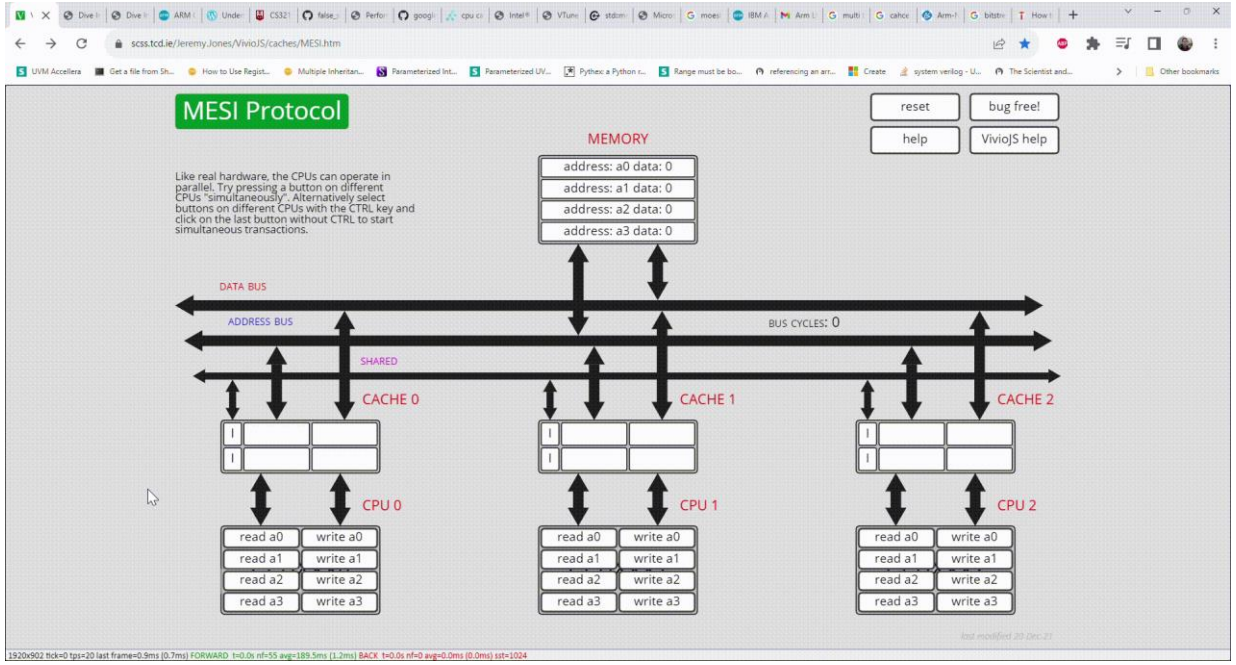


Propuesta 2 (SW $ Coherence *modelling and evaluation)*

Características

1- El sistema tiene que poseer al menos 3 *PEs*
2- *PEs* 3 instrucciones: `write addr reg`, `read addr reg`, `incr reg`
3- Memoria 32 bits datos 16 *entries* mínimo
4- *$* 4 *entries* de 32 bits
5- $ implementa ***write-back*** *policy*
6- Ejecución debe ser paralela (mediante uso de ***threads***)
8- Cada PE debe permitir stepping/start para procesar instrucciones
9- La interfaz gráfica debe permitir generar código aleatorio para cada procesador, y elegir el protocolo a simular (MOESI, MESI, etc)
10- Al final de la simulación el sistema debe presentar un resumen de total de transacciones y su tipo (INV, READ_REQ/RESP, WRITE_REQ/RESP)
11- El sistema permite visualizar los estados de las líneas de cache

Extra
12- Implementar al menos una métrica adicional para evaluar el desempeño (modelando tiempo de ejecución por instrucción, *hit/miss time*, energía consumida por *bus cycles*)



https://www.scss.tcd.ie/Jeremy.Jones/VivioJS/caches/MESI.htm

Propuesta 2 (SW $ Coherence *modelling and evaluation)*

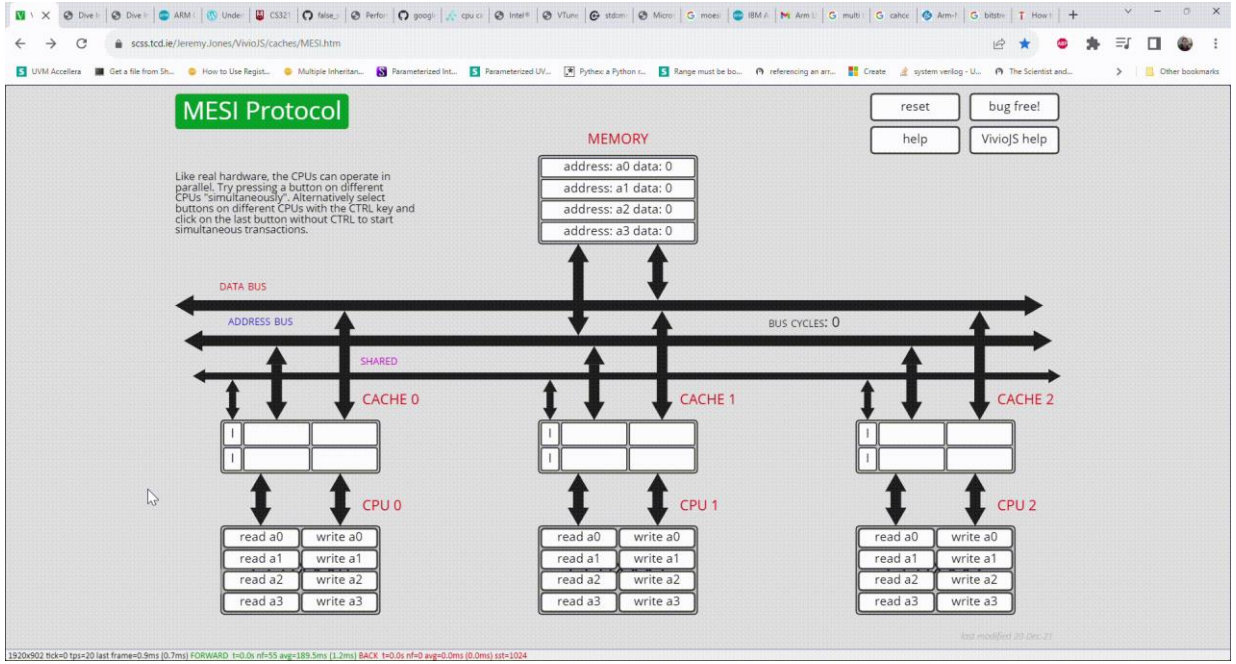
Entregables

1- Propuesta de diseño (evaluación ventajas/desventajas)
2- Especificación de componentes del sistema (*backend*, *frontend*)
3- Explicación proceso de diseño
4- Demostración funcional
5- Evaluación de desempeño de los 2 protocolos
6- Manual de usuario
7- Reporte de resultados

Extra

8- Plan de pruebas y cobertura de funcionalidades*
9- Perfilado y análisis de desempeño del sistema

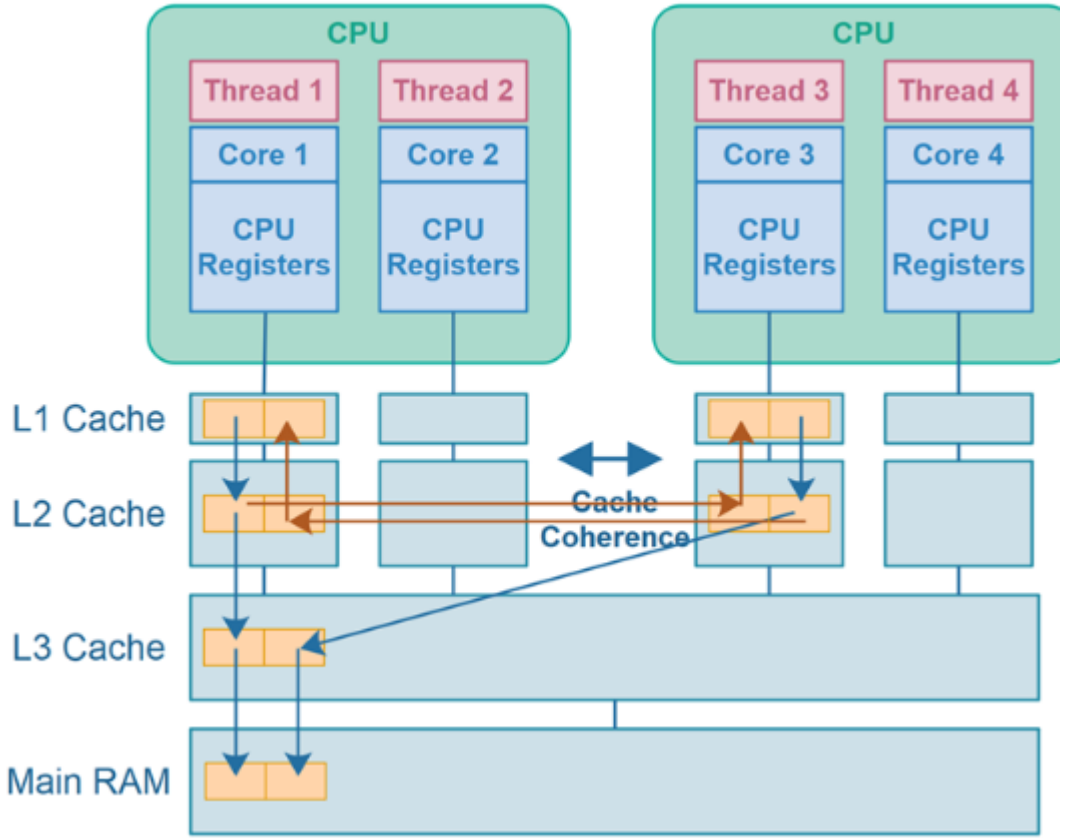
*Si su grupo es de 4 integrantes el punto 8 es obligatorio



https://www.scss.tcd.ie/Jeremy.Jones/VivioJS/caches/MESI.htm

Propuesta 3 (*MP Programming and false sharing)*

- *Caches* son fundamentales para mejorar el desempeño de un *PE*
- *Cache Coherence* permite el manejo correcto de memoria compartida
- Al implementar SW que emplee paralelismo (MT) es necesario entender el HW objetivo y el efecto de $ coherence en el desempeño



Propuesta 3 (*MP Programming and false sharing*)

Objetivos

1- Investigar el concepto de *false sharing* y su efecto en el desempeño de un sistema *MP*

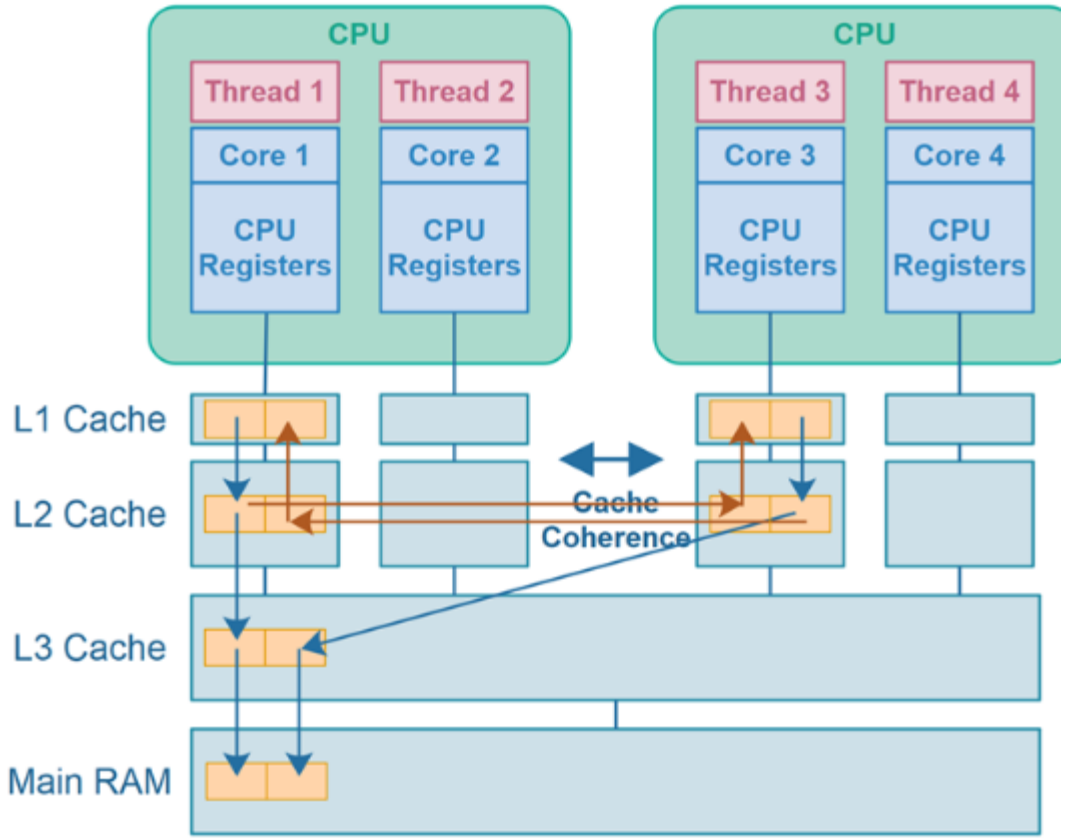
2- Crear pruebas de exploración que permitan observar el efecto de *false sharing*

3- Utilizar herramientas de perfilado y/o análisis exploratorio como (*perf, vtune, cachegrind*, gem5***)

4- Evaluar el fenómeno false sharing en al menos 2 lenguajes de programación

5- Analizar las características de HW objetivo a la hora de implementar SW

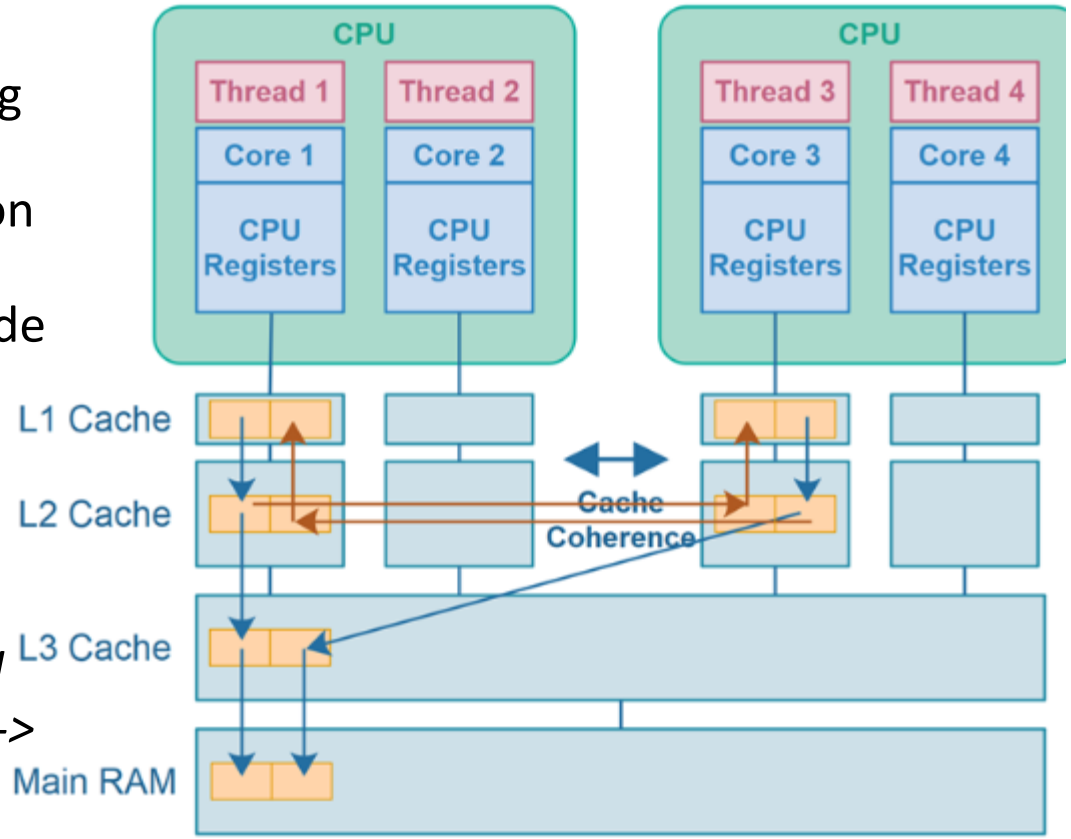
6- Implementar al menos 2 medidas para mitigar el *false sharing*



Propuesta 3 (*MP Programming and false sharing*)

Características

1- Crear al menos 2 *benchmarks* que implementen multi-threading y single threading (*std:threads, pthreads*, etc) y muestren *false sharing* **en al menos 2 lenguajes de programación** (recomendación C, C++, Rust, Go, C#)*
2- Los *benchmarks* deben ser capaces de determinar parámetros de su HW objetivo (*$ line size, number of cores, threads, $ levels and sizes*) -> *portability*
3- Crear una etapa de automatización que permita correr los benchmarks múltiples veces y capturar sus resultados usando herramientas de perfilado (perf, vtune, etc)
4- Graficar los resultados obtenidos *Single Thread vs Multi-Thread*
5- Implementar **al menos 2 medidas** para mitigar el *false sharing* -> agregarlos a los *benchmark* usar 3 y 4 para generar resultados



***El *benchmark* debe evidenciar al máximo el false sharing**

Extra

6- Emplear *más de una herramienta de perfilado y comparar resultados*
7- Correr los benchmarks en más de un sistema

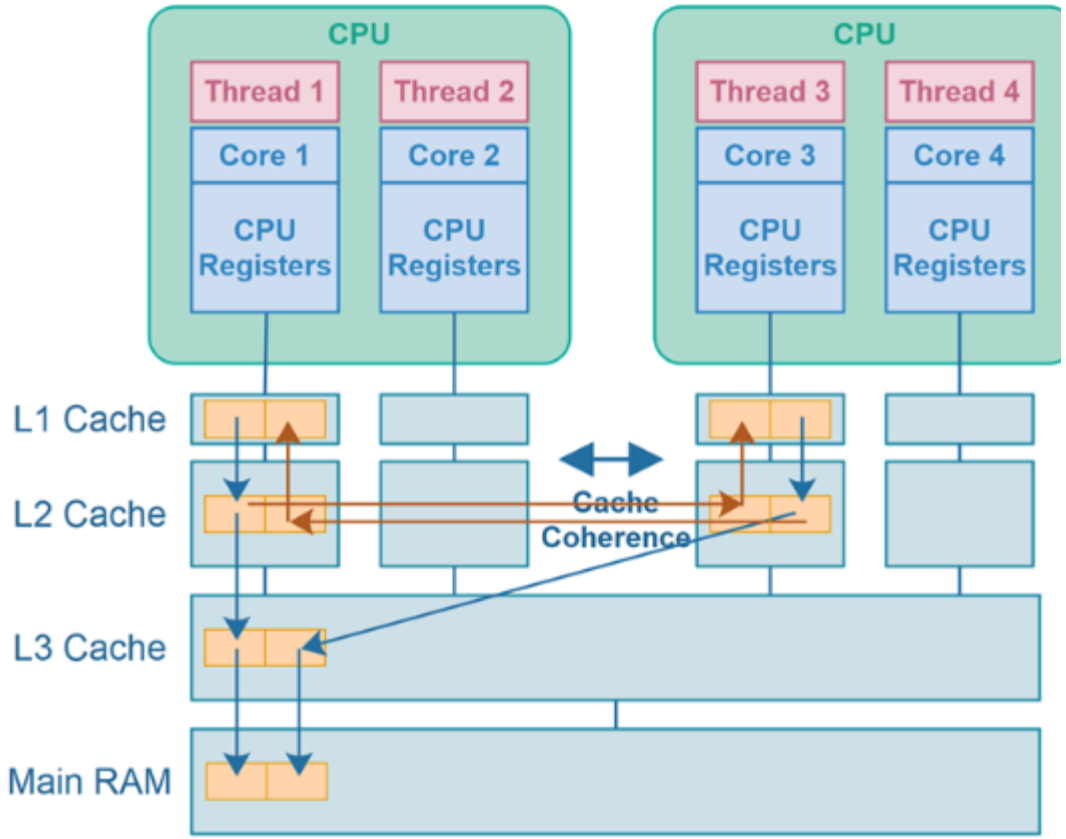
Propuesta 3 (*MP Programming and false sharing*)

Entregables

1- Propuesta de diseño (evaluación ventajas/desventajas)
2- Especificación de componentes del sistema
(*backend*, *frontend, benchmarks, HW detection code*)
3- Explicación proceso de diseño
4- Demostración funcional (básica)
5- Evaluación de desempeño en *ST, MT (w/o false sharing)*
6- Manual de usuario
7- Reporte de resultados

Extra
8- Plan de pruebas y cobertura de funcionalidades*
9- Auto detección de *false sharing* y solución semi-automática* + demostración
11- Evaluación con diferentes protocolos usando Ruby y gem5**

*Si su grupo es de 4 integrantes el punto 8 y 9 son obligatorios
** Únicamente para un *benchmark* y *C++*



Referencias

1- https://en.wikipedia.org/wiki/False_sharing

2- https://www.gem5.org/documentation/learning_gem5/part3/MSIintro/

3- Hennessy, J., & Patterson, D. (2017). Computer Architecture: A Quantitative Approach (6th ed.). Elsevier Science.

4- Sorin, J & Hil, Mark D. (2011) A Primer on Memory Consistency and Cache Coherence.

5- https://www.scss.tcd.ie/Jeremy.Jones/VivioJS/caches/MESI.htm