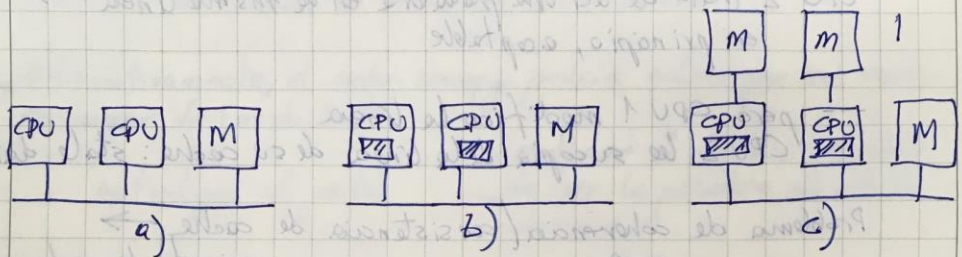
Fecha / Date: 14 nov. 297

UMA Symmetric Multiproc. Arch.

un único bus



a) Si la CPU quiere leer una palabra de memoria:
   pruebe a ver si el bus está ocupado
   si está desocupado, pone la dirección de la palabra en el bus
      pone en 1 algunas señales de control
      espera hasta que la memoria ponga la palabra
   si está ocupado, espera hasta que esté desocupado

   3 o 4 CPUs: la competencia por el bus es manejable;
   32 o 64: insoportable; la mayoría de las CPUs estarán descupadas la mayor parte del tiempo

b) Agregamos un cache a c/ CPU (dentro o al lado):
   para muchos ~~reads~~ reads basta el cache → menos tráfico en el bus → más CPUs
   ¿consistencia de los cachés?

c) Cada CPU tiene cache y memoria local con bus privado:
   - el compilador debe poner programa, strings, constantes y otros datos solo de lectura, stacks y variables locales en memoria privada
   - memoria compartida solo para variables compartidas
   - reduce significativamente el tráfico en el bus

Notas / Notes:

RHEIN

Fecha / Date:

Cache "snooping" (fisgón) husmear
"snoopy" curioso

CPU 1 tiene una línea en su cache
CPU 2 trata de leer una palabra en la misma línea → en principio, aceptable

--- pero CPU 1 modifica la línea
CPU 2 lee su copia de la línea de su cache: stale data

Problema de coherencia / consistencia de cache →
Protocolos (de) para coherencia de cache → conjunto de reglas que previenan que diferentes versiones de la misma línea aparezcan simultáneamente en dos o más caches

Controlador de cache diseñado para "escuchar" el bus:
monitorea todas las solicitudes el bus de las otras CPUs y caches, y en algunos casos hace algo
"snoops"

Protocolo write through: la memoria está siempre actualizada
todos los writes → la palabra va hasta la memoria

Acción	Solicitud local (cache 1) (solicitud el bus)	Solicitud remota → snooper (cache 2)
Read miss	lee dato de la memoria	no hace nada
Read hit	usa dato de cache local	no se entera
Write miss	actualiza dato en memoria	no mira a ver si tiene la palabra
Write hit	actualiza cache y memoria	invalida cache: remote miss

la línea NO es cargada al cache

solicitud de write en el bus

marca la línea en su cache como invalida: elimina el dato del cache

Notas / Notes:

RHEIN

Fecha / Date:

Todos los caches "snoop" todas las solicitudes el bus:
cuando una palabra es escrita,
se actualiza en el cache originario
se actualiza en la memoria
se elimina de todos los otros cachés

Alternativamente, el cache snoopy podría actualizar su cache en lugar de invalidarlo

actualizar el cache

invalidar el cache, seguido de leer la palabra desde la memoria

conceptualmente, lo mismo

variaciones

Protocolos eligan entre estrategia de actualización
estrategia de invalidación

se comportan diferentemente frente a cargas distintas:
los mensajes de actualización son más largos que las invalidaciones, pero previenen cache misses

También se puede cargar el cache "snoopy" en los write misses
solo afecta el desempeño, no la corrección
¿cuál es la probabilidad de que una palabra recién escrita vuelva a ser escrita luego?
- Alta → cargar cache cuando hay write misses → política write-allocate
- Baja → mejor no actualizar; si se lee luego, va a ser cargada de todas maneras (read miss)

Solución simple... pero ineficiente: todo write va a la memoria a través del bus → cuello de botella
Otros protocolos: No todos los writes van a la memoria; se pone un bit en el cache para indicar que la memoria está desactualizada → protocolo write-back

Notas / Notes:

RHEIN

Fecha / Date:

Protocolo write-back MESI (Core i7) de 4 estados:

modificado: entrada válida; memoria inválida; no hay copias

exclusivo: ningún otro cache tiene la línea; memoria está actualizada

"shared": varios caches pueden tener la línea; memoria está actualizada

inválido: la entrada en el cache NO contiene datos válidos

Al inicio del sistema, todas las entradas de los caches son I

la primera lectura de una línea de la memoria a un cache, la entrada es E

siguientes lecturas de esa CPU van al cache (no usan el bus)

Otra CPU^2 lee la misma línea a su cache; la anterior lo ve y anuncia en el bus que también tiene una copia: ambas S

CPU^2 escribe en su cache la línea S: pone una señal de invalidación en el bus para que las otras CPUs se deshagan de sus copias. La línea cambiada pasa a M, pero NO es escrita en la memoria

(una línea E es escrita, no es necesario invalidar nada)

CPU^3 lee la línea a su cache; CPU^2 sabe que la copia en memoria no está actualizada → pone señal en el bus para que CPU^3 espere hasta que CPU^2 escriba la línea en memoria
Solo entonces CPU^3 lee y la línea es S

Si CPU^2 vuelve a escribir la línea, le marca M e... invalida copia de CPU^3

CPU^1 escribe la línea → ...

Notas / Notes:

RHEIN

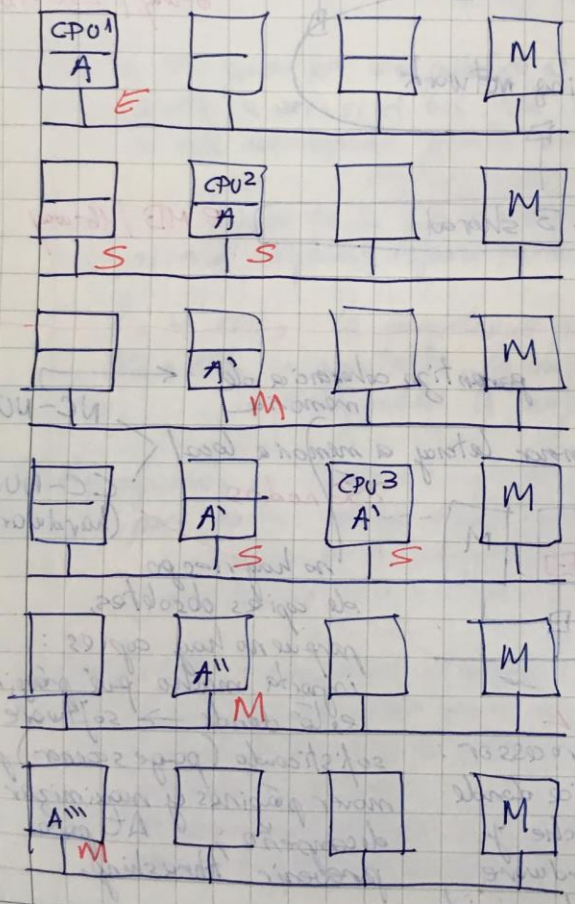
Fecha / Date:

→ - CPU 2 lo ve y pone señal en el bus para que CPU 1 espere hasta que escriba la línea en la memoria; entonces pone su propia copia en I, ya que sabe que CPU 1 la va a cambiar

CPU 1 está ~~implementada~~ escribiendo una línea que no está en su caché:

write-allocate ⟹ la línea se carga en la caché y se marca M

de lo contrario, la escritura va directo a la memoria



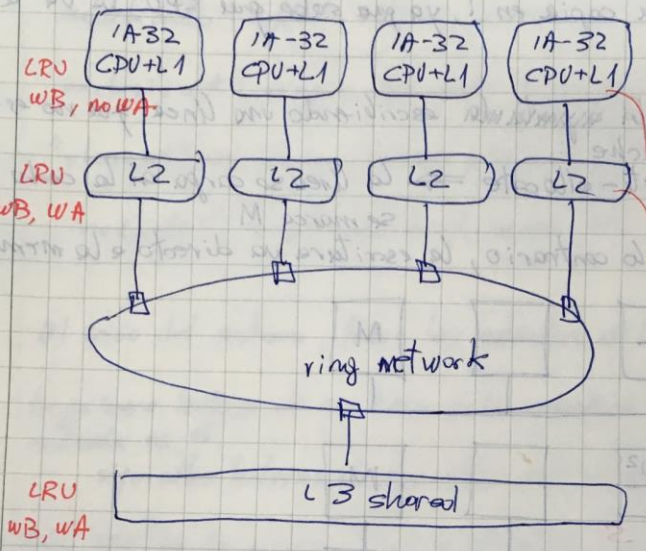
Notas / Notes:

RHEIN

Fecha / Date:

Core i7 : single-chip multiprocessor

direcciones virtuales de 48 bits ; físicas, de 44



(registros, 64bits)

32 kB (4-way instrs. (8-way datos)

8-way / 256 kB

8 MB / 16-way

UMA

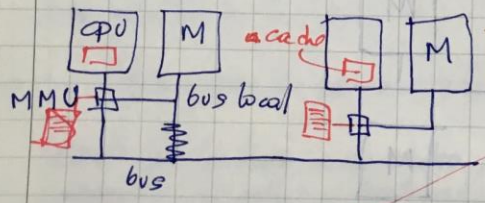
NUMA → escalan, menor latency a memoria local

garantiza coherencia de memoria ←

NC-NUMA

CC-NUMA (hardware DSM)

256 nodos



no hay riesgo de copias obsoletas, porque no hay copias : importa mucho qué página está dónde → software sofisticado (page scanner) para mover páginas y maximizar desempeño ; AT para prevenir thrashing

directory-based multiprocessor : una base de datos que dice dónde está cada línea del caché y cuál es su estado : hardware especial extremadamente rápido

Notas / Notes: (fracción de un ciclo del bus)

RHEIN