Paper Review:

Efficient Virtual Memory for Big Memory Servers

Jiale Zhang

Noviembre 2023

Contenido

Problemática

Se observó que:

- Las cargas de trabajo en **'Big Memory' servers** enfrentan problemas con la memoria virtual basada en páginas.
- Hasta un 10 % de ciclos gastados en TLB misses.
- Estas, rara vez necesitan o usan la flexibilidad y ventajas que da la memoria basada en páginas.

Cita

"Virtual memory was invented in a time of scarcity. Is it still a good idea?"
– Charles Thacker, 2010 ACM Turing Award Lecture.

Precedentes

- La memoria virtual basada en páginas no ha cambiado de manera sustancial desde los finales de los 60, cuando se introdujeron las TLBs.
- Una motivación histórica era la capacidad de aprovechar al máximo la escasa memoria física sin intervención del programador.
- Hoy en día esto no es un problema y la memoria virtual basada en páginas tiene un coste elevado en *big memory servers*.
- Las cargas de trabajo de estos *big memory servers* no suelen usar ni swapping, ni copy-on-write ni per-page-protection que ofrecen las páginas.

Propuesta

Se propone:

- Usar *direct segment* para mapear parte de memoria virtual contigua de un proceso a memoria contigua física.
- Para ello se necesita HW adicional. La introducción de los registros Base, Limit y Offset por core.
- Elimina TLB misses en estructuras de datos clave como buffer pools o key-value stores en memoria.

Se prototipa soporte SW para el *direct segment* para la arquitectura x86-64 y se emula el HW necesario con buenos resultados.

Parecido con Segmentación

¿No se parece mucho al uso de la segmentación en memoria?

Diferencias con Segmentación

¿No se parece mucho al uso de la segmentación en memoria?

No, a diferencia de la segmentación tiene tres diferencias clave:

1. Mantiene un espacio de direcciones virtual lineal.
2. No trabaja sobre paginado.
3. Puede coexistir con páginas de otras direcciones virtuales.

Contenido

Máquina de Test

	Description
Processor	Dual-socket Intel Xeon E5-2430 (Sandy Bridge), 6 cores/socket, 2 threads/core, 2.2 GHz
L1 DTLB	4KB pages: 64-entry, 4-way associative; 2MB pages: 32-entry 4-way associative; 1GB pages: 4-entry fully associative
L1 ITLB	4KB pages: 128-entry, 4-way associative; 2MB pages: 8-entry, fully associative
L2 TLB (D/I)	4 KB pages: 512-entry, 4-way associative
Memory	96 GB DDR3 1066MHz
OS	Linux (kernel version 2.6.32)

Figura: Características del equipo donde se han ejecutado las pruebas

Workloads Empleados

graph500	Generation, compression and breadth-first search of large graphs. http://www.graph500.org/
memcached	In-memory key-value cache widely used by large websites (e.g., Facebook).
MySQL	MySQL with InnoDB storage engine running TPC-C (2000 warehouses).
NPB/BT NPB/CG	HPC benchmarks from NAS Parallel Benchmark Suite. http:// nas.nasa.gov/publications/npb.html
GUPS	Random access benchmark defined by the High Performance Computing Challenge. http://www.sandia.gov/~sjplimp/algorithms.html

Figura: Descripción de los workloads empleados

Uso de Memoria Virtual

- **Swapping**. La cantidad de memoria ha sido escogida cuidadosamente para que las cargas de trabajo tengan suficiente capacidad.x Se comprueba con *vmstat*.
- **Memory Allocation and Fragmentation**. Los Big Memory workloads suelen reservar la mayoría de la memoría al comenzar la aplicación y se mantiene estable.
- **Per-Page Permissions**. Más del 99 % de la memoría reservada se hace con permisos de lectura y escritura.

Uso de Memoria Virtual

Reserva de memoria de big memory workloads

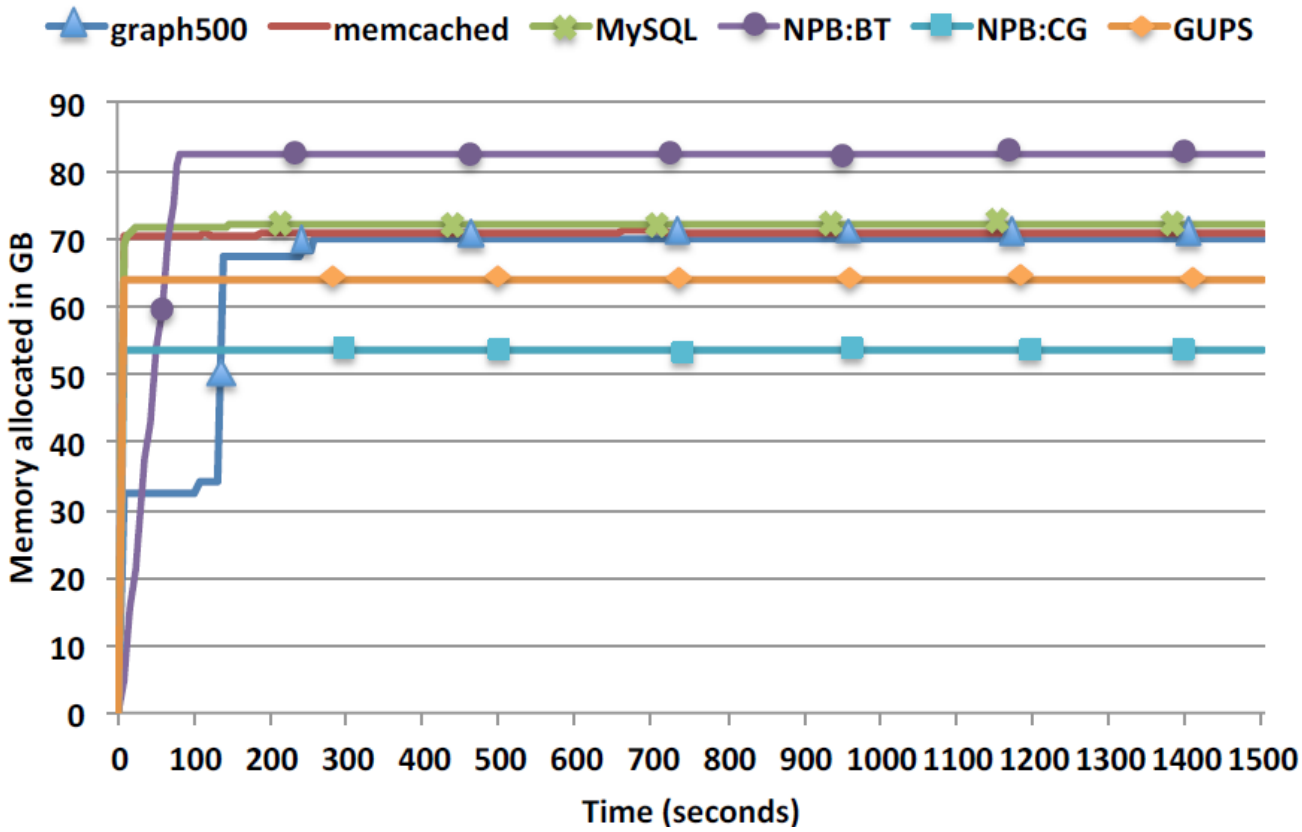

Figura: Observamos que la reserva de memoria ocurre sobre todo al principio del programa

Uso de Memoria Virtual

Permisos de Lectura/Escritura

	Percentage of allocated memory with read-write permission
graph500	99.96%
memcached	99.38%
MySQL	99.94%
NPB/BT	99.97%
NPB/CG	99.97%
GUPS	99.98%

Figura: Porcentaje de páginas con permisos de lectura/escritura

Coste de Memoria Virtual

- Aumentar tamaño de TLB aumenta latencia.
- Se pueden emplear diferentes tamaños de página.
- Se analiza el número de ciclos gastados por TLB misses en el equipo de test.

Coste de Memoria Virtual

Coste TLB miss

	Percentage of execution cycles servicing TLB misses			
	Base Pages (4KB)		Large Pages (2MB)	Huge Pages (1GB)
	D-TLB	I-TLB	D-TLB	D-TLB
graph500	51.1	0	9.9	1.5
memcached	10.3	0.1	6.4	4.1
MySQL	6.0	2.5	4.9	4.3
NPB:BT	5.1	0.0	1.2	0.06
NPB:CG	30.2	0.0	1.4	7.1
GUPS	83.1	0.0	53.2	18.3

Figura: Coste de TLB miss en los workloads

Conclusiones de Big Memory Workloads

- Son programas de larga duración. Encendidos 24x7.
- Las caches de memoria y las bases de datos son configuradas para adaptarse a los recursos disponibles.
- Suelen tener unos pocos procesos primarios que consumen la mayor parte de la memoria.

Contenido

Soporte HW

- Mapear memoria física a partir de un *direct segment* usando tres registros adicionales
- Se usa paginación en direcciones fuera de este rango.

Soporte HW

Traducción direcciones con *sirect-segment*

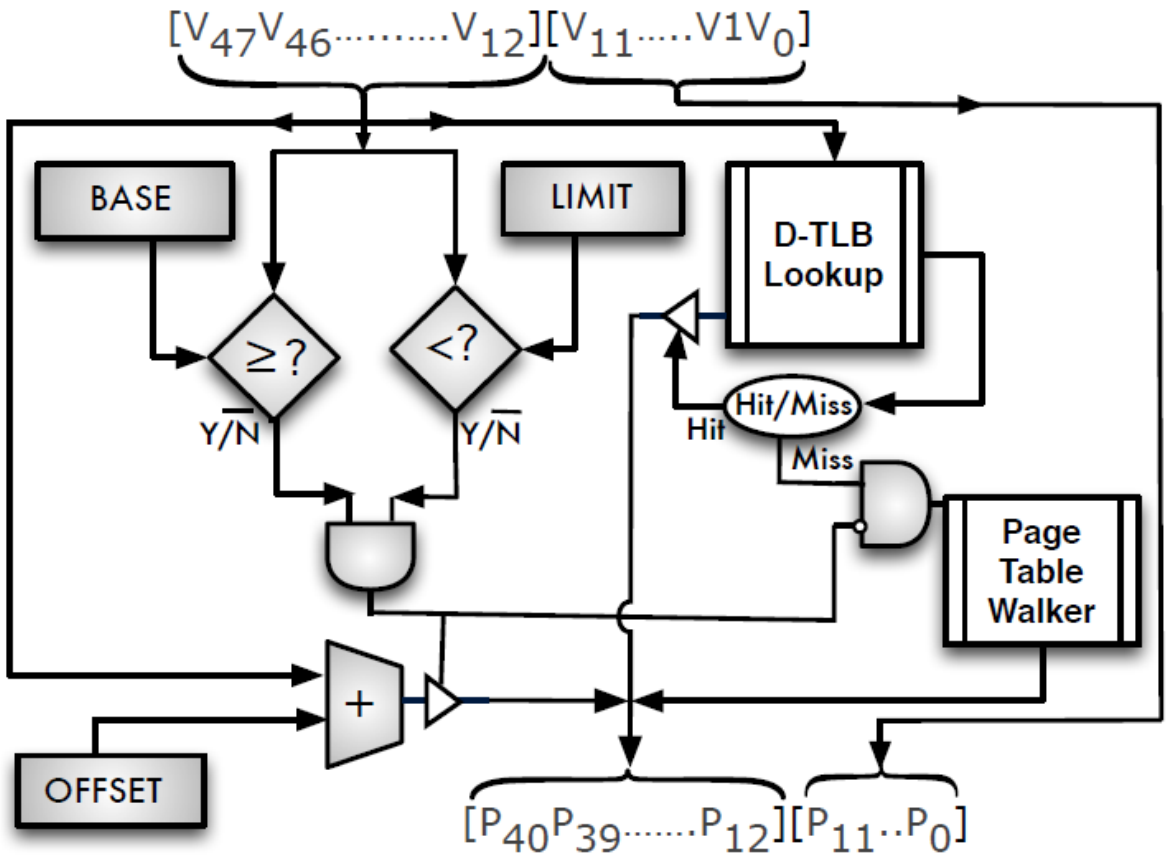

Figura: Vista lógica de la traducción de direcciones con *direct-segment*

Soporte HW

Los *direct-segments* NO...

El *direct-segment* no cuenta con las siguiente capacidades y características:

- No exporta direcciones de memoria bidimensionales a aplicaciones.
- No reemplaza la paginación.
- No trabaja por encima de la paginación.

Soporte SW

El software del sistema se encarga de las siguientes tareas:

- Otorgar la abstracción de *primary region* a las aplicaciones para que especifiquen la región de memoria elegida.
- Se encarga de dar la memoria física correspondiente a la *primary region*.
- Gestionar los registros del *direct-segment*.
- Aumentar y disminuir el tamaño del *direct-segment*.

Soporte SW

Memoria virtual con *primary region*

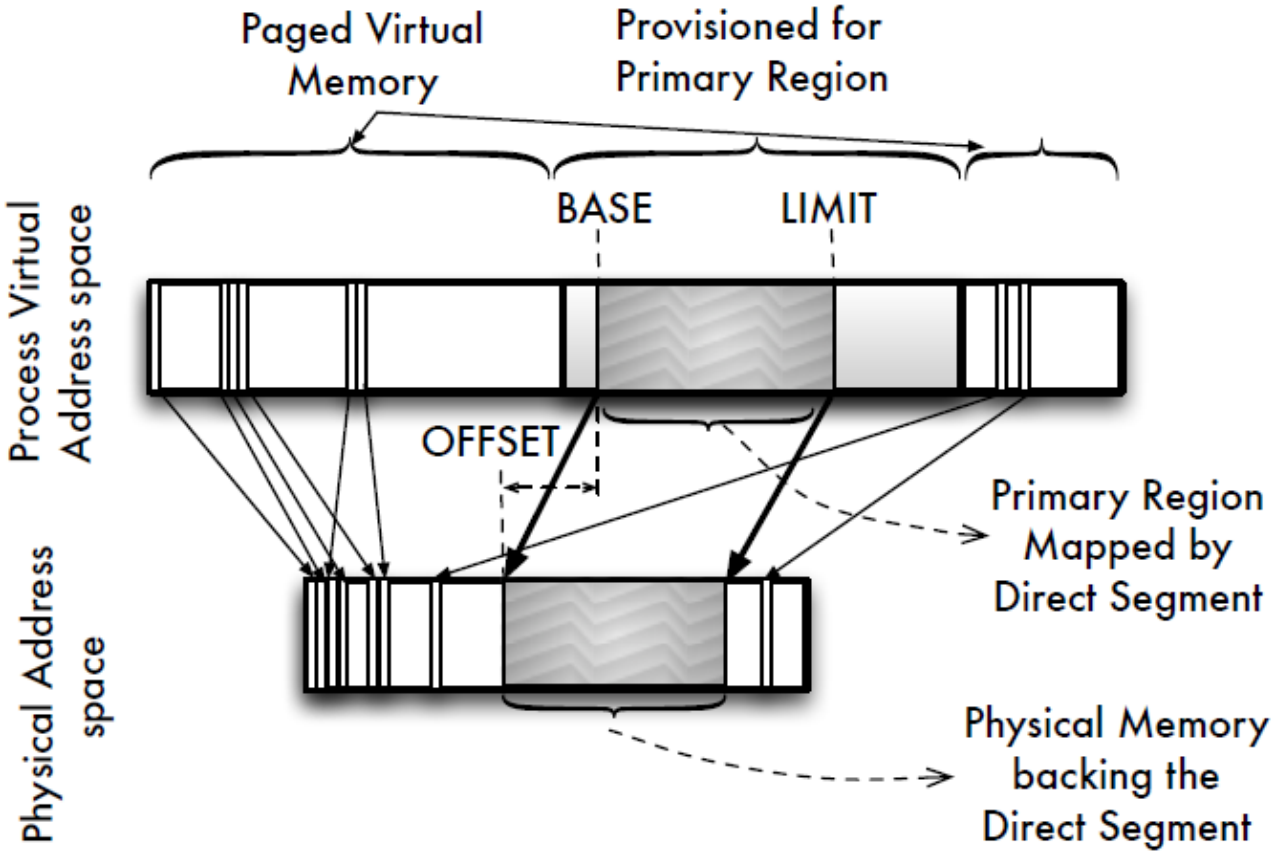

Figura: Direcciones virtuales y físicas con *primary region*

Contenido

Prototipo SW

Se implementa el prototipo modificando el kernel de Linux 2.5.32 (x86-64). Este código tiene dos partes:

- Implementación de abstracción de *primary region*, independiente de la arquitectura. Se encarga de proveer de memoria física y asignar la *primary region*.
- Código específico de cada arquitectura para instanciar *primary region* y modelar *direct segment*.

Prototipo SW

Implementación independiente de la arquitectura

- Simplificado asumiendo que solo un proceso puede usar el *direct segment* en un momento determinado de tiempo.
- Se ha implementado un nuevo system call para **indentificar el primary process**. Este system call también avisa al SO de la identidad del proceso y la cantidad de memoria estimada para la *primary region*.
- Toda la memoria anónima se guarda con permisos de escritura/lectura en la *primary region*.
- La memoria física se reserva con Linux *memory hotplug*. Esto se hace al iniciarse el *primary process*.

Prototipo SW

Implementación dependiente de la arquitectura

- Sin HW real, se emulan las funcionalidades del *direct segment* con páginas de 4KB.
- Se modifica el *page fault handler* para que calcule la dirección física a partir del número de la página virtual que ha fallado.

Metodología

- Simulaciones de sistemas completas de *big memory servers* pueden llevar meses y una capacidad de memoria muy grande.
- Para resolver esto se usan HW performance counters, cogiendo datos con *oprofile*, y se modifica el kernel para que un TLB miss se convierta en un fallo de página 'falso' y compruebe si este se encuentra en el rango del *primary segment*.
- Para forzar un trap existen una serie de bits reservados en las PTE. Si cambiamos el bit **present** de x86-64 podemos generar un *trap* cada vez que se intente cargar información.

Metodología

Explicación detallada

- Cuando un PTE en memoria cambia este no es actualizado automáticamente, por lo que una entrada del TLB puede seguir siendo usada a pesar de que haya cambiado el bit reservado.
- Se marcan todas las PTE de una *primary region* como inválidas. Al acceder a una dirección se actualiza la entrada del TLB con el PTE indicado. En este momento cambiamos el bit present del PTE provocando inconsistencia. Al intentar un re-fetch y producirse un TLB miss, este lanzará una excepción con un código único.
- Guardamos la dirección y comprobamos si está en el rango de la *primary region*. Volvemos a cargar el PTE en el TLB y envenenamos el PTE de memoria.

Resultados

Porcentaje ciclos por TLB miss

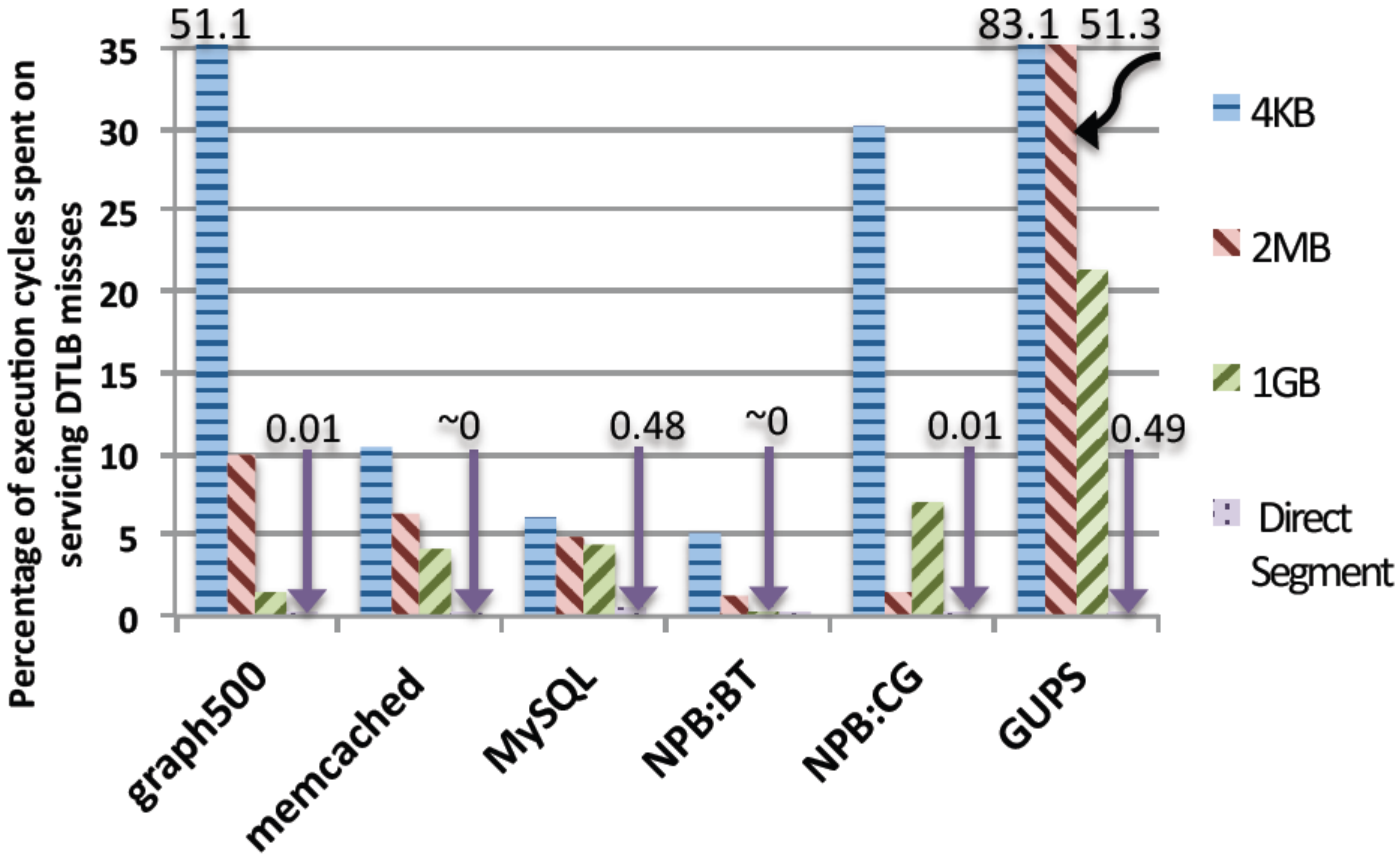

Figura: Porcentaje de ciclos gastados en DTLB misses

Resultados

Porcentaje TLB miss en direct segment

	Percent of D-TLB misses in the direct secgment
graph500	99.99
memcached	99.99
mySQL	92.40
NBP:BT	99.95
NBP:CG	99.98
GUPS	99.99

Figura: Porcentaje de reducción de TLB misses

Resultados

Escalado de GUPS

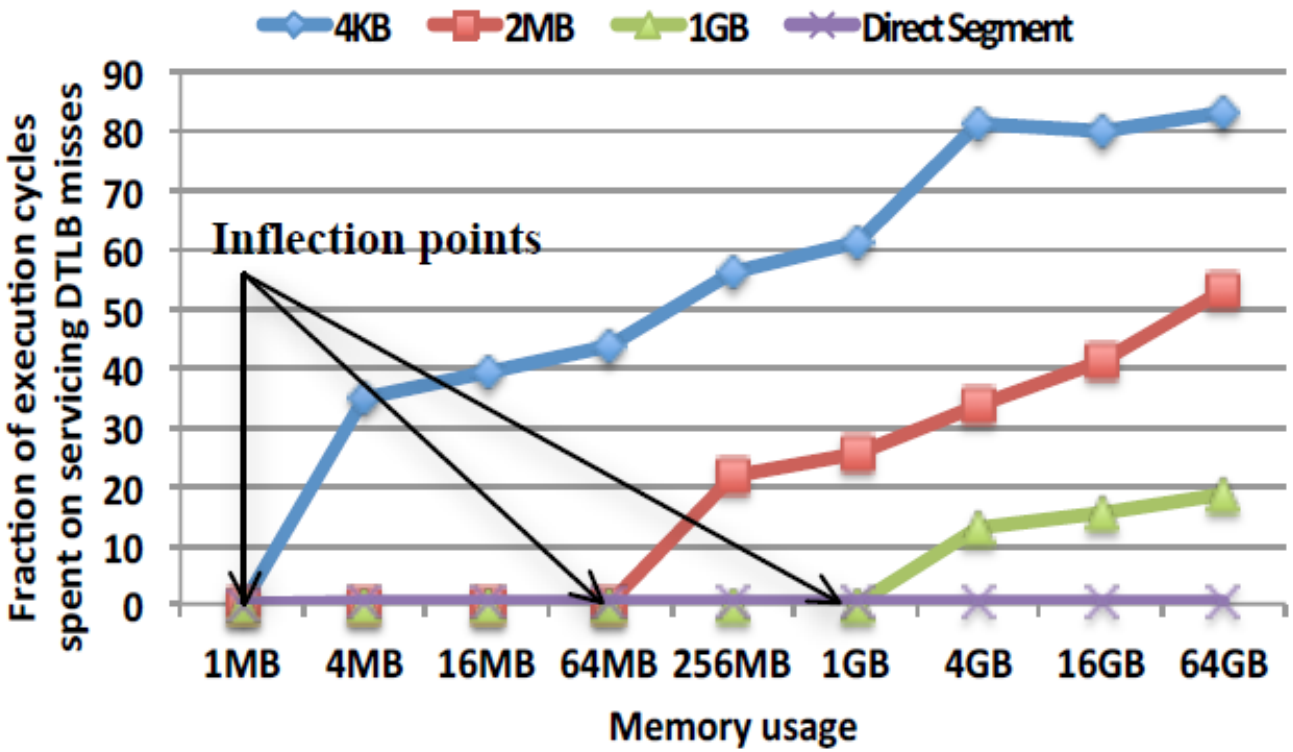

Figura: Evolución TLB miss al escalar GUPS

Contenido

Evaluación

Puntos fuertes

- Diseño sencillo y 'fácil' de implementar.
- Resultados bastante buenos.
- Explicación concisa de metodología.
- Consciente de los puntos débiles.

Críticas

- Demasiado sencillo, dirigido a problemas específicos.
- Generación de traps por SW provoca ruido en resultados.
- El mapeo es de cierta manera subjetivo.
- Hay alguna errata.

Efficient Virtual Memory for Big Memory Servers

TIEMPO DE PREGUNTAS