Università degli studi di Roma Tor Vergata
Facoltà di Ingegneria

Trasferimenti di Dati tra Memoria Utente e Kernel tramite Circuiti DMA Dedicati

Candidato:
Antonio Papa

Relatore:
Prof. Marco Cesati

Correlatore:
Ing. Emiliano Betti

A.A. 2012-2013

Schema della presentazione

Le Copie in Memoria

- Le copie in memoria sono implementate attraverso una serie di operazioni di **load** e **store** effettuate dalla CPU.

- Le istruzioni di **load** e **store** coinvolgono una serie di risorse tra cui:
  - I registri della CPU (32 o 64 bit)
  - La memoria cache

Le Copie in Memoria

- Le copie in memoria sono implementate attraverso una serie di operazioni di **load** e **store** effettuate dalla CPU.

- Le istruzioni di **load** e **store** coinvolgono una serie di risorse tra cui:
  - I registri della CPU (32 o 64 bit)
  - La memoria cache

Inoltre si può incorrere in fenomeni come la **Cache Pollution**

Cache Pollution

I dati di un'applicazione vengono caricati in cache inutilmente provocando la rimozione di preziose risorse precedentemente memorizzate.

Copie memoria-memoria tra spazio Kernel e utente

- Ogni chiamata di sistema effettua operazioni di copia tra memoria kernel e utente

Copie memoria-memoria tra spazio Kernel e utente

- Ogni chiamata di sistema effettua operazioni di copia tra memoria kernel e utente

- Sono utilizzate dai driver per copiare dati dai/nei buffer dello spazio utente tramite *file operation*

  Alcuni scenari in cui vengono utilizzate:

  - elaborazioni video
  - acquisizione dati da sensori

Copie memoria-memoria tra spazio Kernel e utente

- Ogni chiamata di sistema effettua operazioni di copia tra memoria kernel e utente

- Sono utilizzate dai driver per copiare dati dai/nei buffer dello spazio utente tramite *file operation*

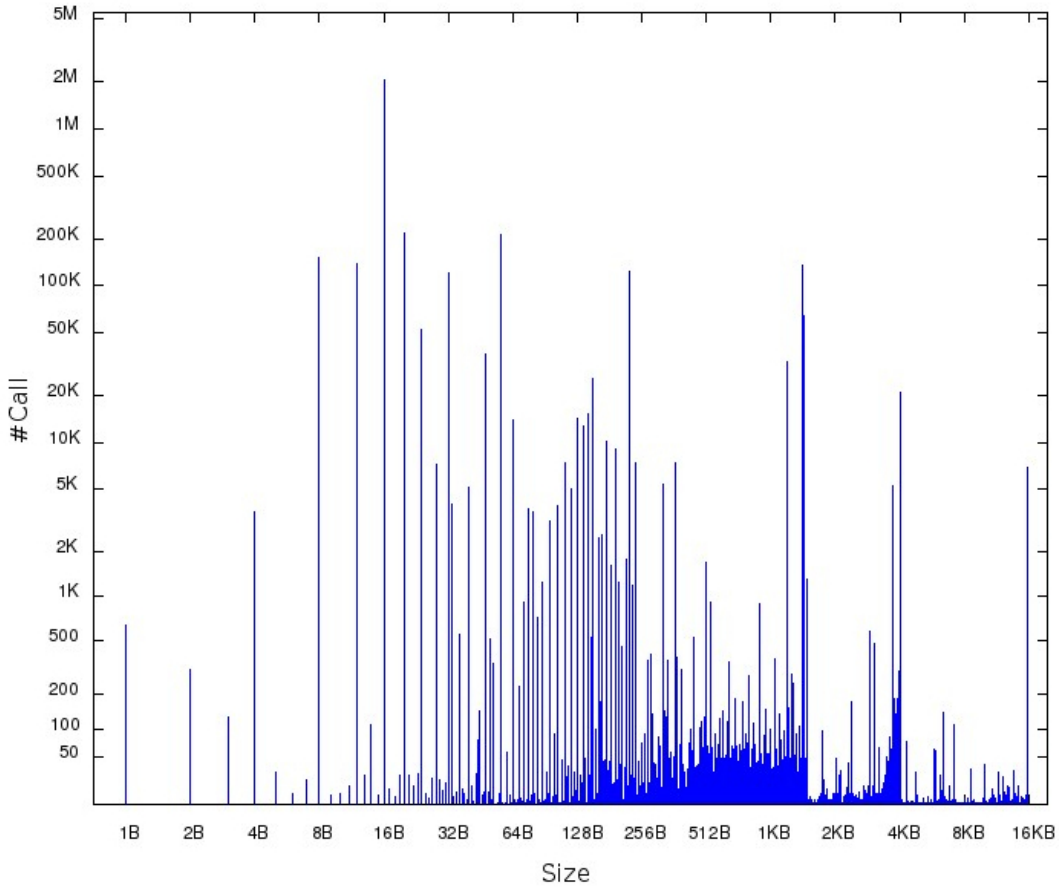
  Alcuni scenari in cui vengono utilizzate:
  - elaborazioni video
  - acquisizione dati da sensori

- Inoltre i trasferimenti di dati tra memoria Kernel e utente sono utilizzati in diversi sottosistemi del Kernel

  Ad esempio:
  - Lo stack TCP/IP copia i payload dei pacchetti di rete (*receive-side*) dallo spazio di memoria Kernel a quello dell'applicazione utente

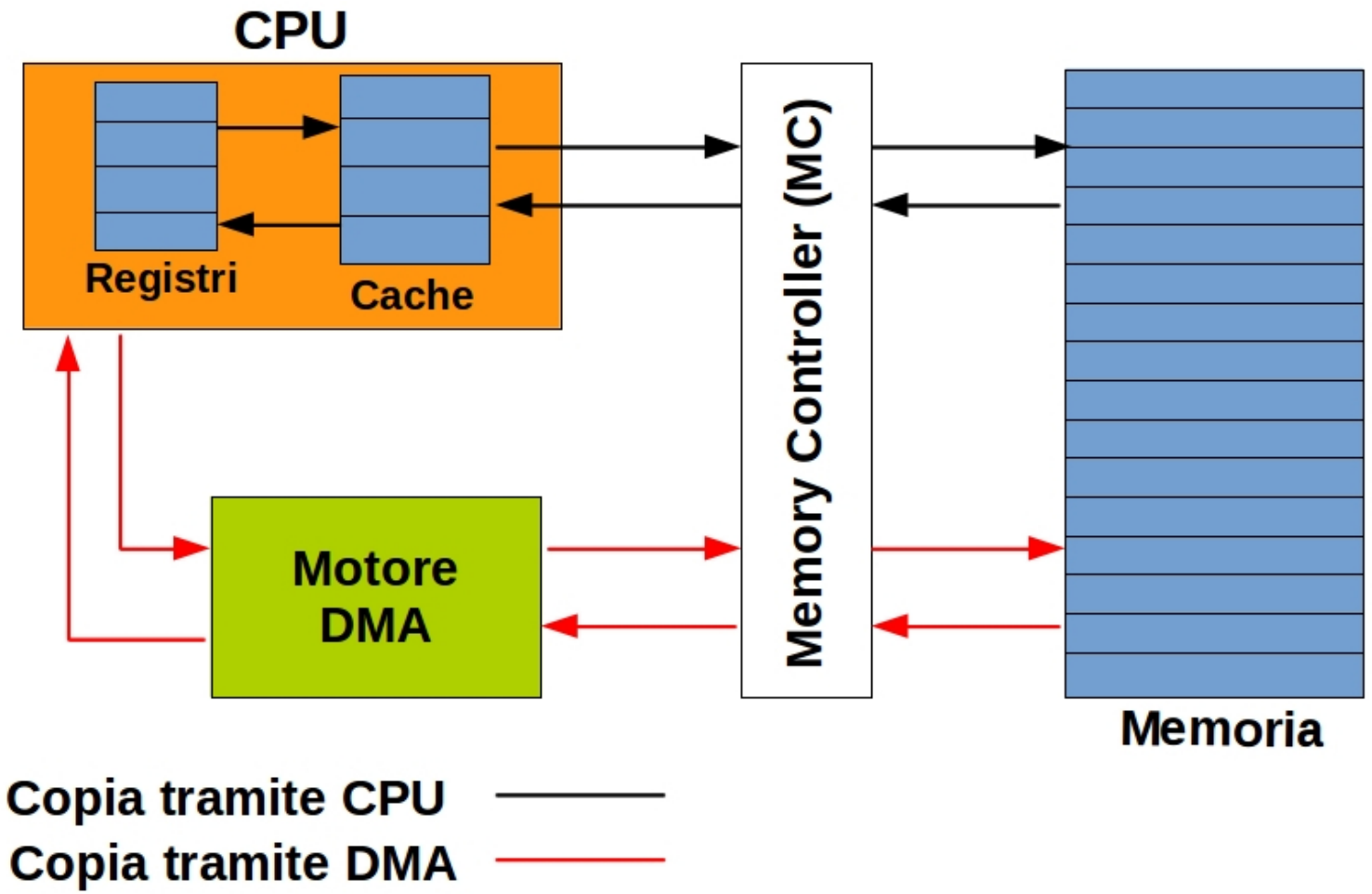
Trasferimenti di dati tra memoria Kernel e utente

- S.O. sottoposto ad un medio carico di lavoro
- Intervallo di tempo di 100 secondi



~ 3,6 M operazioni di copia tra memoria kernel e utente

Perché un Motore Hardware per le copie in memoria?



Copie memoria-memoria tramite motore DMA

Benefici:

- Ottimizzazione delle risorse della CPU
- Offload delle operazioni di copia tra spazio utente e Kernel

Copie memoria-memoria tramite motore DMA

Benefici:

- Ottimizzazione delle risorse della CPU
- Offload delle operazioni di copia tra spazio utente e Kernel

Obiettivi:

- Migliori prestazioni nei trasferimenti di dati
- Predicibilità delle operazioni di copia
- Riduzione della Cache Pollution

Copie memoria-memoria tramite motore DMA

Benefici:

- Ottimizzazione delle risorse della CPU
- Offload delle operazioni di copia tra spazio utente e Kernel

Obiettivi:

- Migliori prestazioni nei trasferimenti di dati
- Predicibilità delle operazioni di copia
- Riduzione della Cache Pollution

Realizzato attraverso un motore DMA indipendente (programmabile), che è molto diffuso nei sistemi embedded:

- *Zynq-7000*, DMA Controller con 8 canali
- *Altera Cyclone V*, DMA Controller con 8 canali

L'Estensione DMA-CFTU (1)

Supporto alle copie in memoria

- Interfaccia generica per i trasferimenti DMA
- Modifica delle funzioni di copia del Kernel
- Due Politiche per l'assegnazione dei canali DMA:
  - *Exclusive Channel*: ogni canale DMA è associato in modo esclusivo ad una CPU (private channel)
  - *Shared Channel*: ogni Channel DMA è condiviso (public channel)

L'Estensione DMA-CFTU (1)

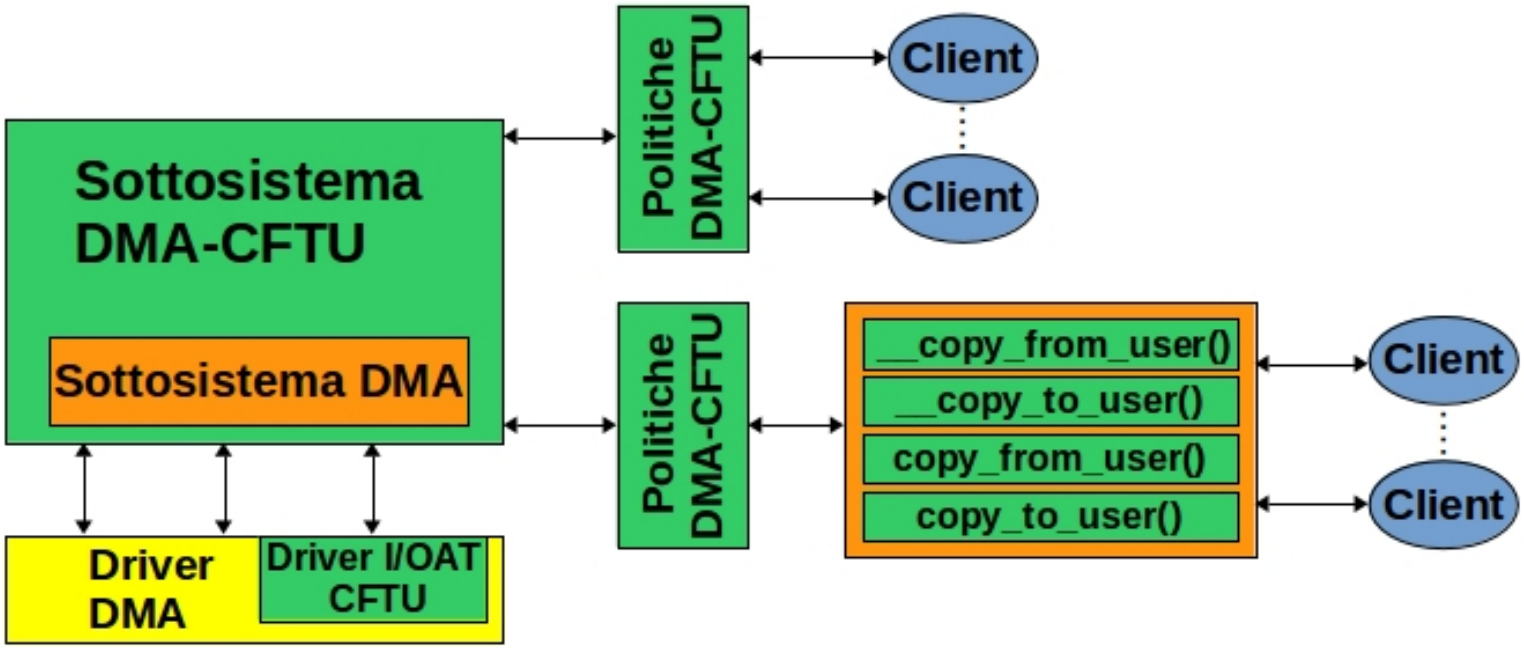
Supporto alle copie in memoria

- Interfaccia generica per i trasferimenti DMA
- Modifica delle funzioni di copia del Kernel
- Due Politiche per l'assegnazione dei canali DMA:
  - *Exclusive Channel*: ogni canale DMA è associato in modo esclusivo ad una CPU (private channel)
  - *Shared Channel*: ogni Channel DMA è condiviso (public channel)

Supporto per applicazioni Real-Time

- Interfaccia per la creazione di politiche Real-Time
- Politica *Priority Channel*

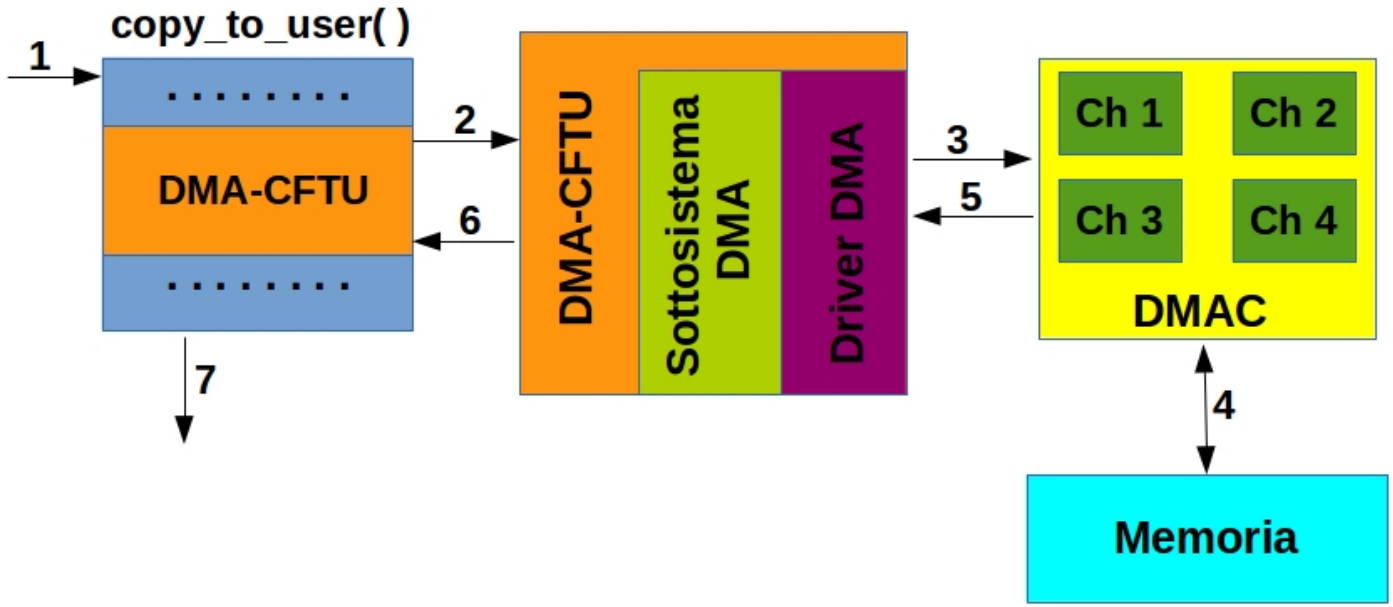
L'Estensione DMA-CFTU (2)



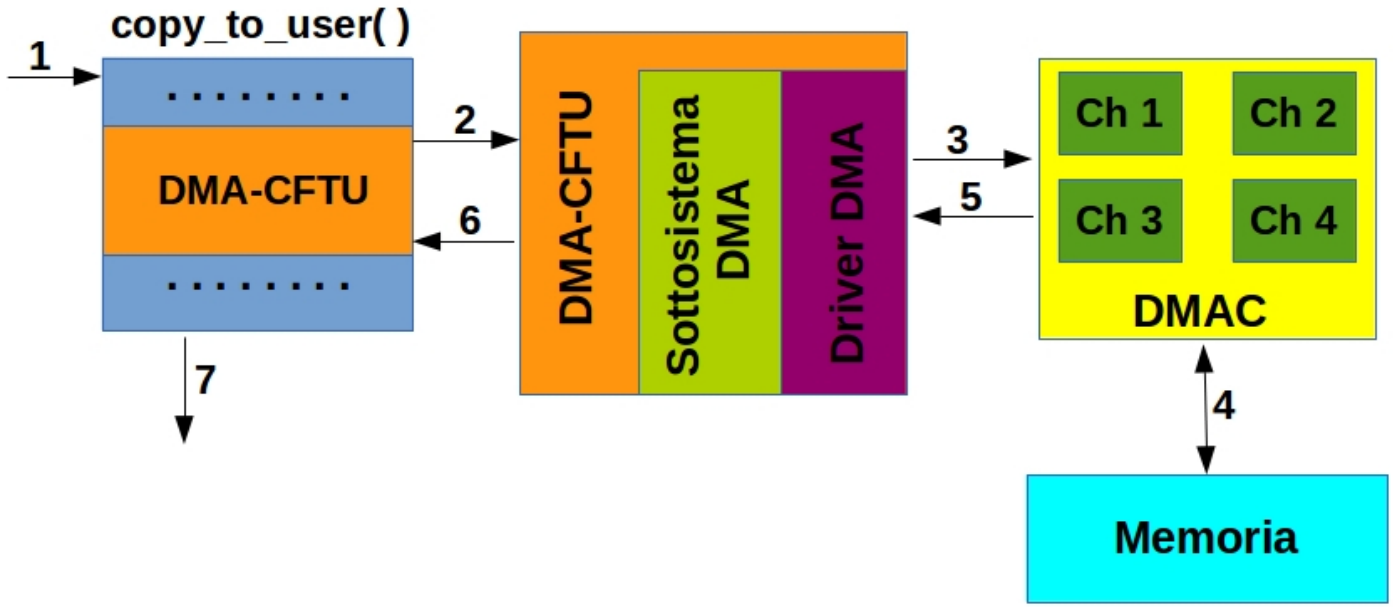
Componenti:

- Driver I/OAT-CFTU
- Sottosistema DMA-CFTU
- Politiche DMA-CFTU

copy_to_user() tramite motore DMA

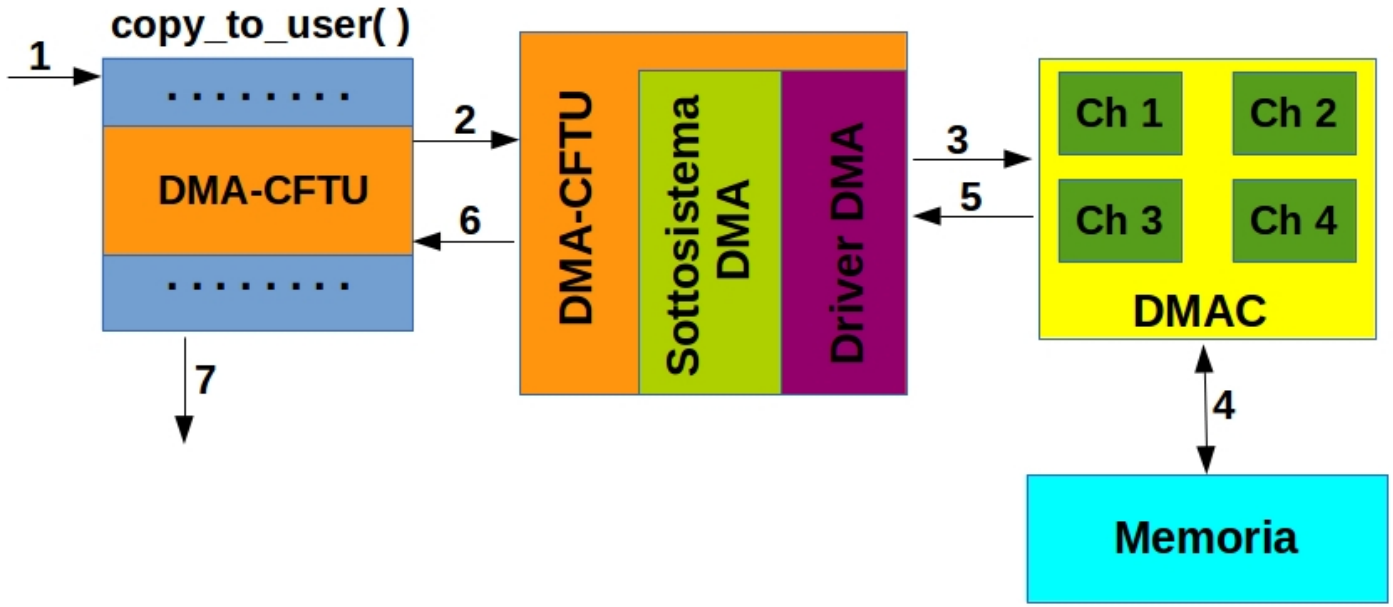


copy_to_user() tramite motore DMA



1 Il Kernel chiama la funzione copy_to_user()

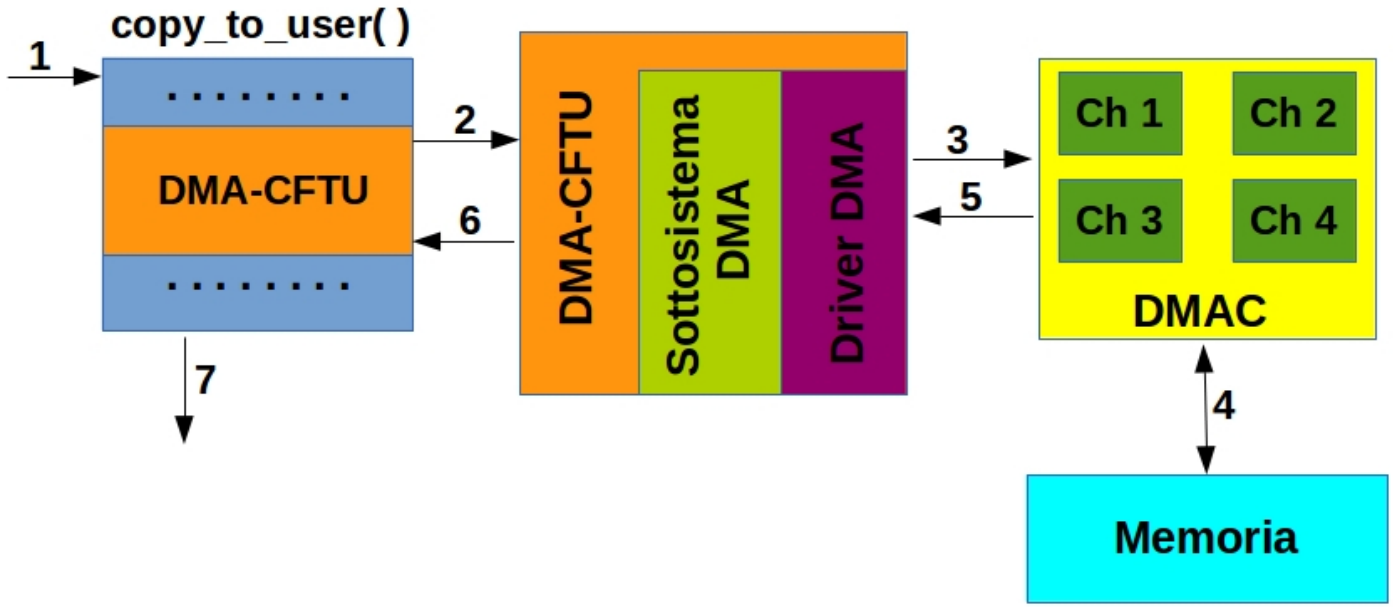
copy_to_user() tramite motore DMA



1 Il Kernel chiama la funzione copy_to_user()

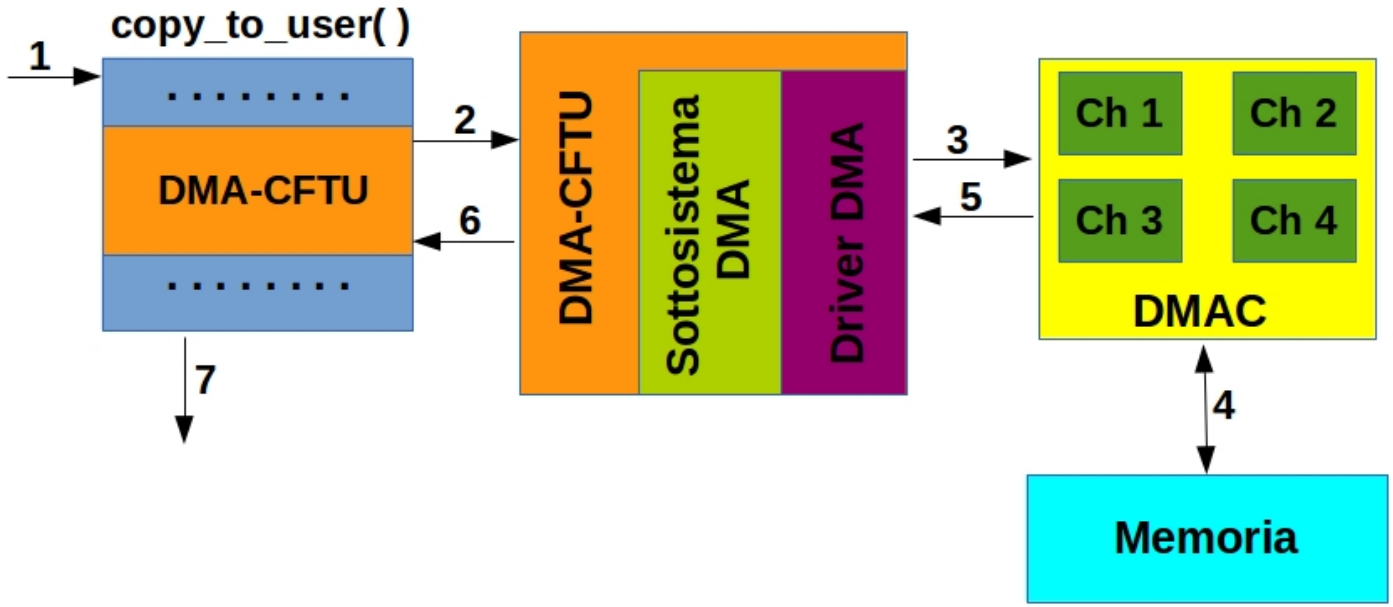
2 L'operazione di copia è delegata all'estensione DMA-CFTU

copy_to_user() tramite motore DMA



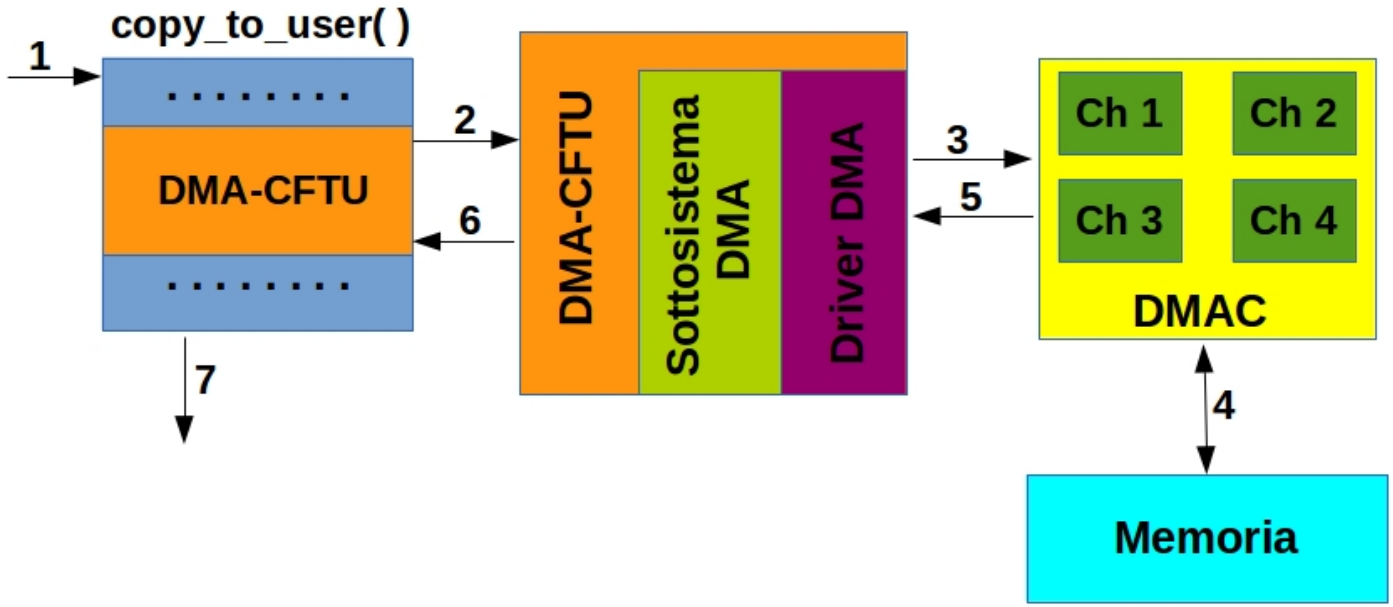
1. Il Kernel chiama la funzione copy_to_user()
2. L'operazione di copia è delegata all'estensione DMA-CFTU
3. Setup del motore DMA

copy_to_user() tramite motore DMA



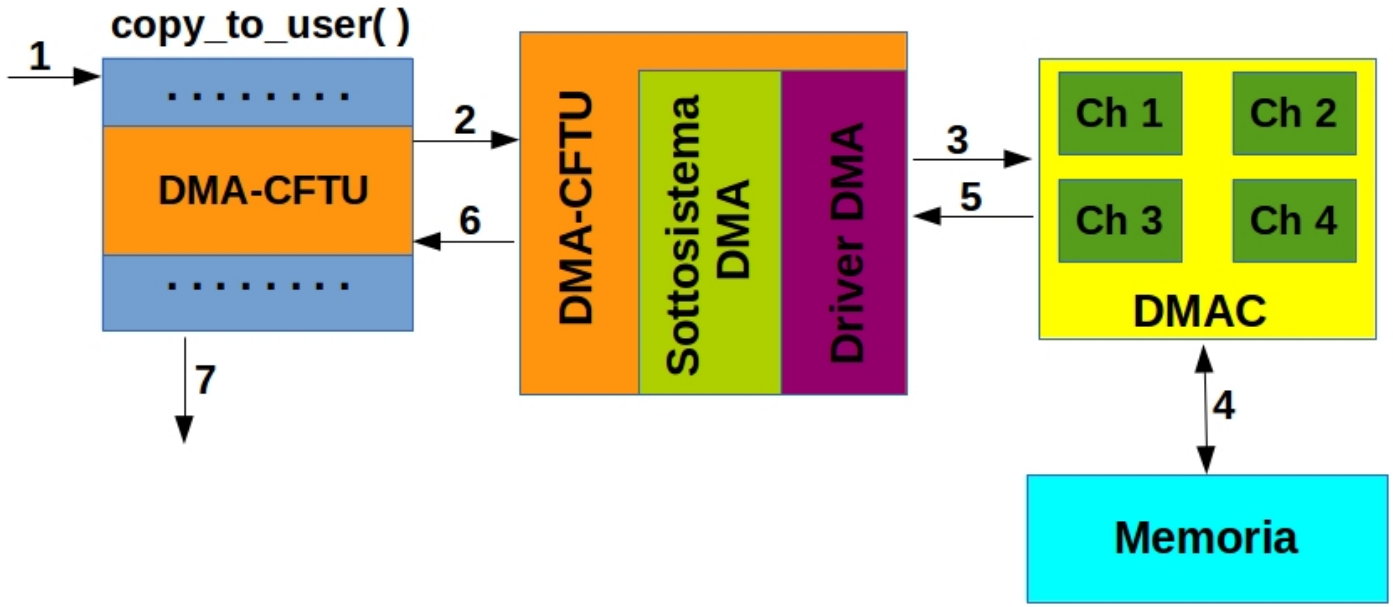
1. Il Kernel chiama la funzione copy_to_user()
2. L'operazione di copia è delegata all'estensione DMA-CFTU
3. Setup del motore DMA
4. L'operazione di copia è effettuata tramite motore DMA

copy_to_user() tramite motore DMA



1. Il Kernel chiama la funzione copy_to_user()
2. L'operazione di copia è delegata all'estensione DMA-CFTU
3. Setup del motore DMA
4. L'operazione di copia è effettuata tramite motore DMA
5. Il motore DMA invia un interrupt

copy_to_user() tramite motore DMA



1 Il Kernel chiama la funzione copy_to_user()
2 L'operazione di copia è delegata all'estensione DMA-CFTU
3 Setup del motore DMA
4 L'operazione di copia è effettuata tramite motore DMA
5 Il motore DMA invia un interrupt
6-7 Ritorno in copy_to_user() e terminazione

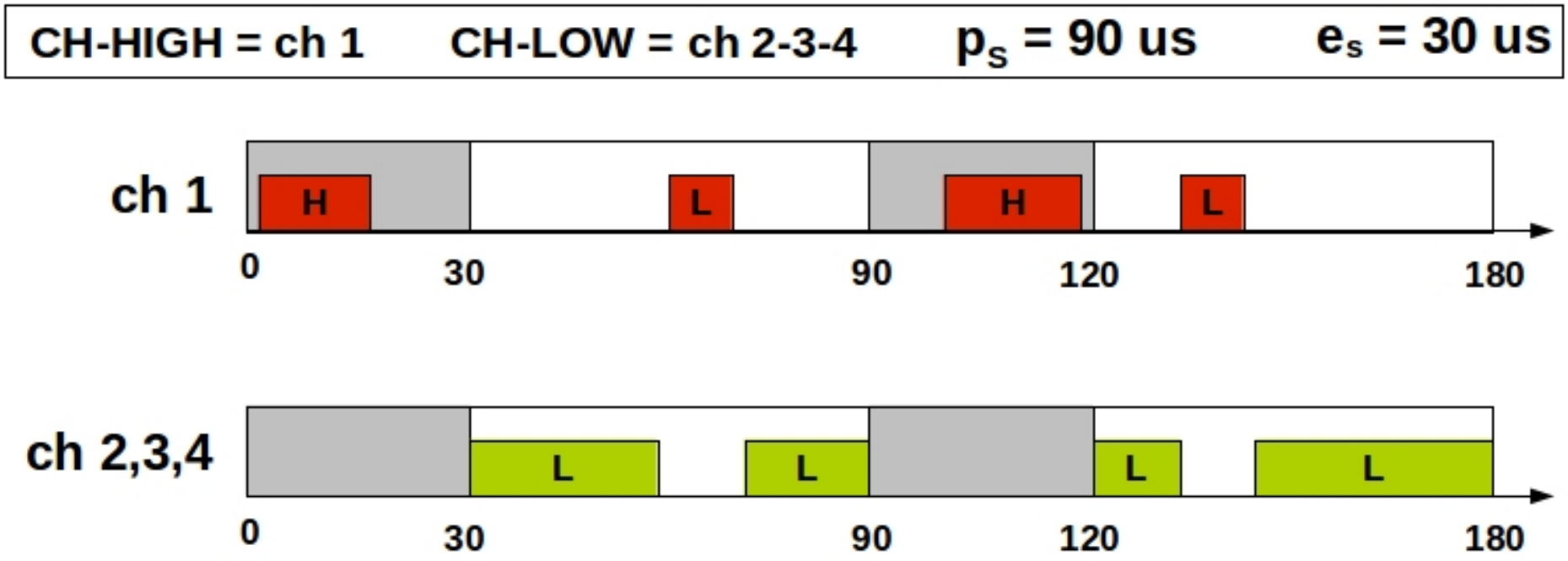
Priority Channel

- I canali *CH-LOW* sono sospesi per un tempo $\mathbf{e}_s$ ogni periodo $\mathbf{p}_s$
- I canali *CH-HIGH* utilizzano il bus DMA in modo esclusivo per un tempo $\mathbf{e}_s$ ogni periodo $\mathbf{p}_s$

Priority Channel

- I canali *CH-LOW* sono sospesi per un tempo $\mathbf{e}_s$ ogni periodo $\mathbf{p}_s$
- I canali *CH-HIGH* utilizzano il bus DMA in modo esclusivo per un tempo $\mathbf{e}_s$ ogni periodo $\mathbf{p}_s$

Ad esempio:



La configurazione di sistema

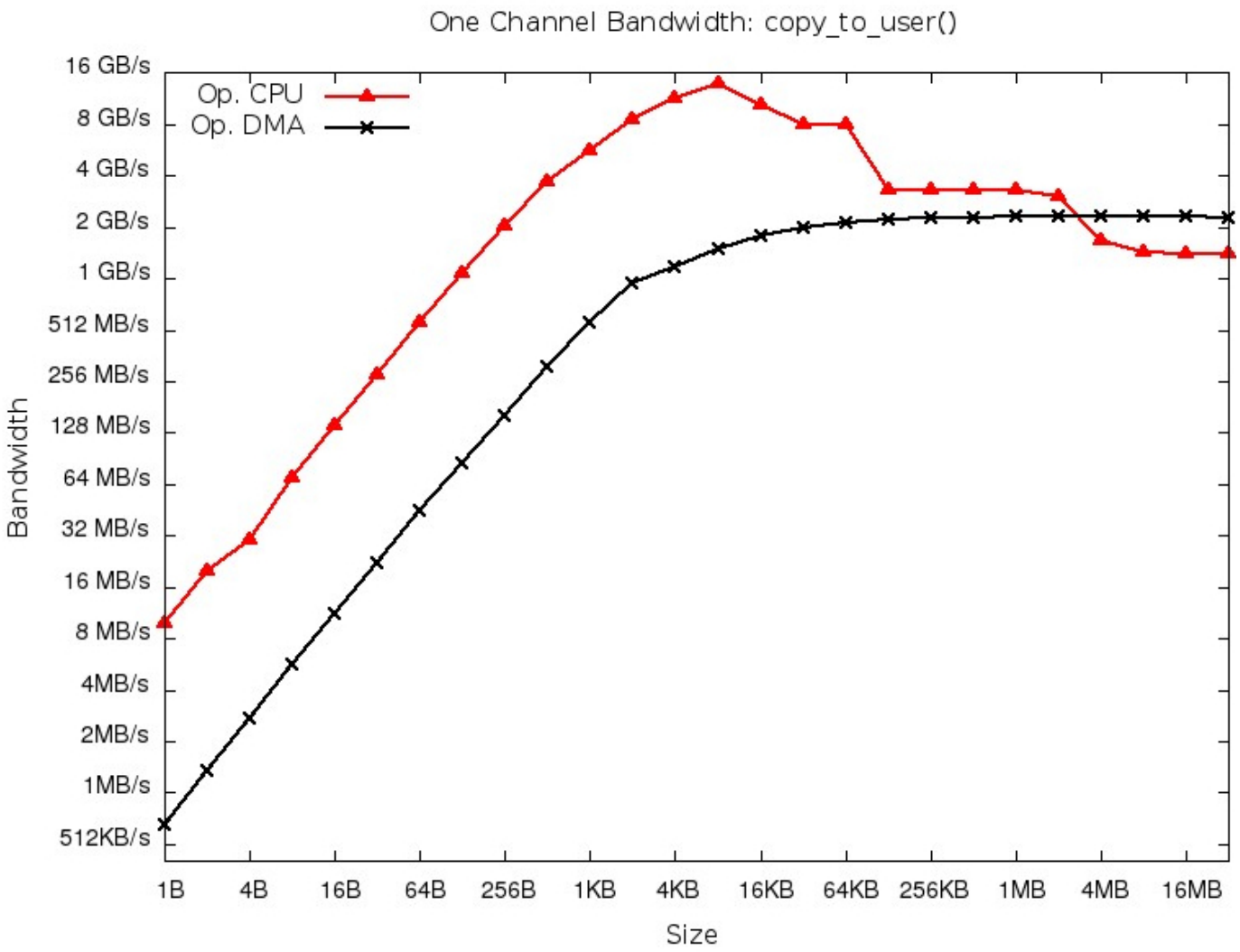
Intel's I/O Acceleration Technology (I/OAT)

- Asynchronous DMA Copy Engine all'interno dell' MCH
- Trasferimenti di dati Memoria-Memoria
- 4 canali DMA

Specifiche di sistema

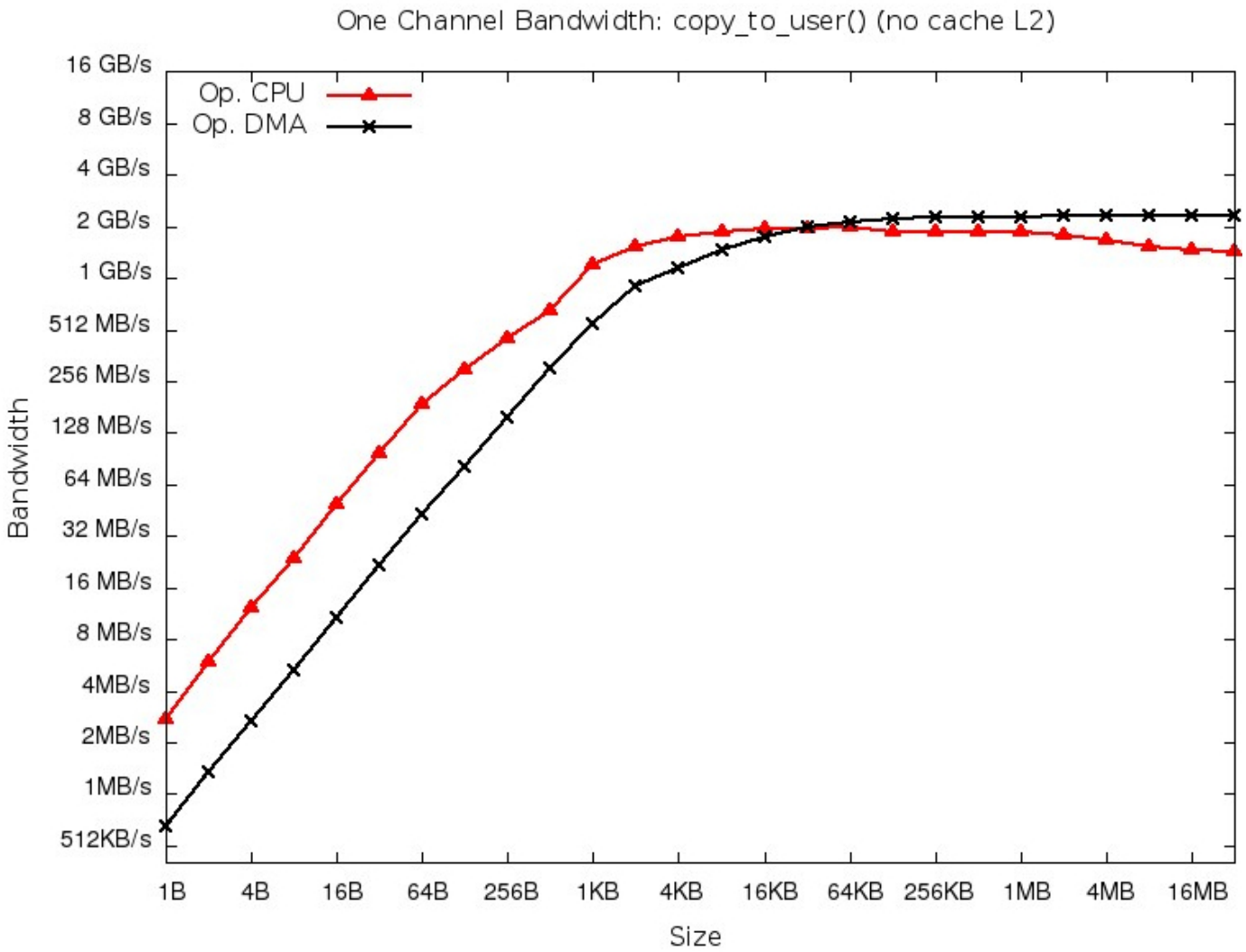
- Intel Xeon 5150 Dual-Core 2.66 GHZ (X2)
- Cache $L1_{d/i}$ 32KB, L2 4MB
- Memoria RAM 4GB
- Intel 5000X Chipset MCH con DMA Engine a 64 bit
- Linux Slackware, Kernel version 3.9.2 (patch CFTU)

One Channel Bandwidth (1)



- Il motore DMA ha prestazioni migliori della CPU su buffer $>$ 2MB

One Channel Bandwidth: (2)

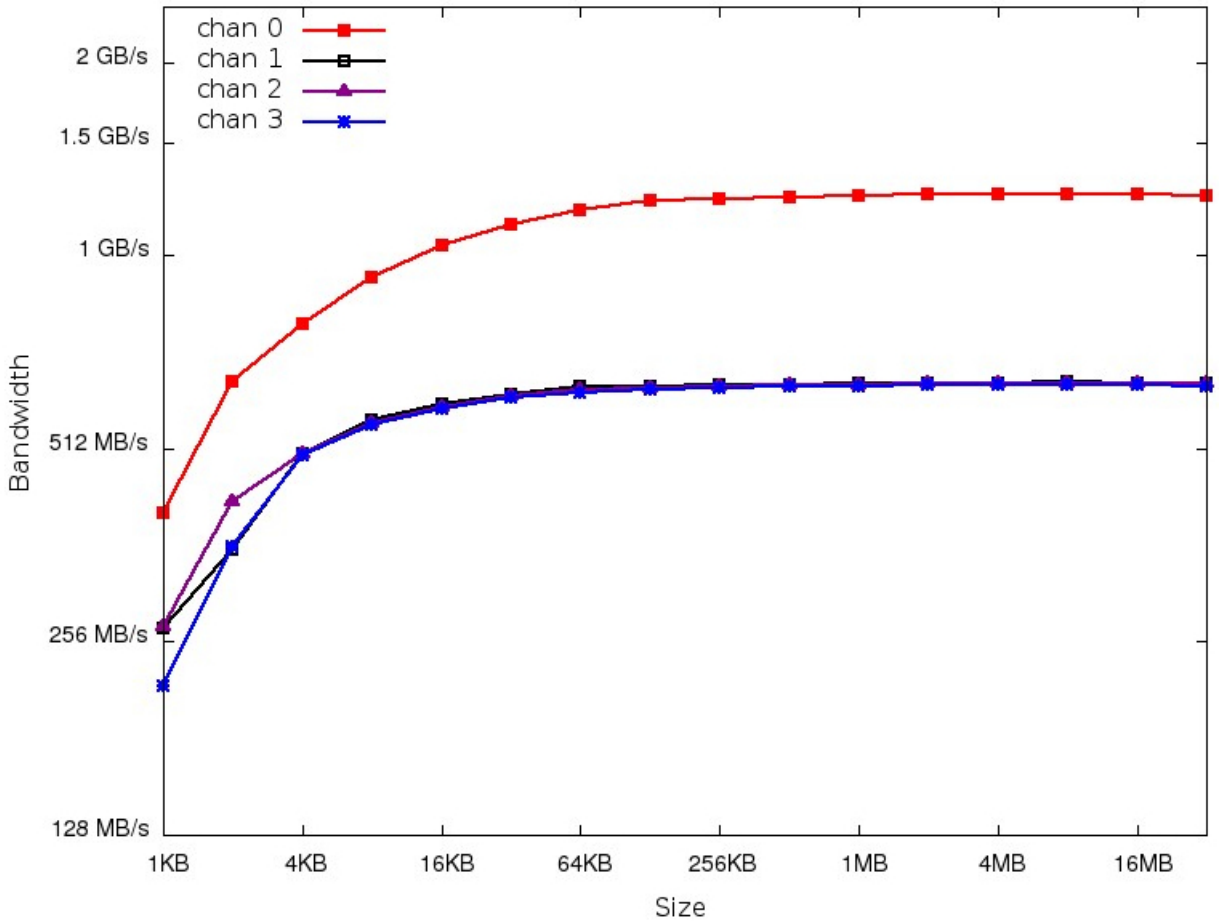


- Il motore DMA ha prestazioni migliori della CPU su buffer $>$ 16KB

Priority Channel Bandwidth

Politica Priority Channel: $p_s = 90\mu s$, $e_s = 30\mu s$

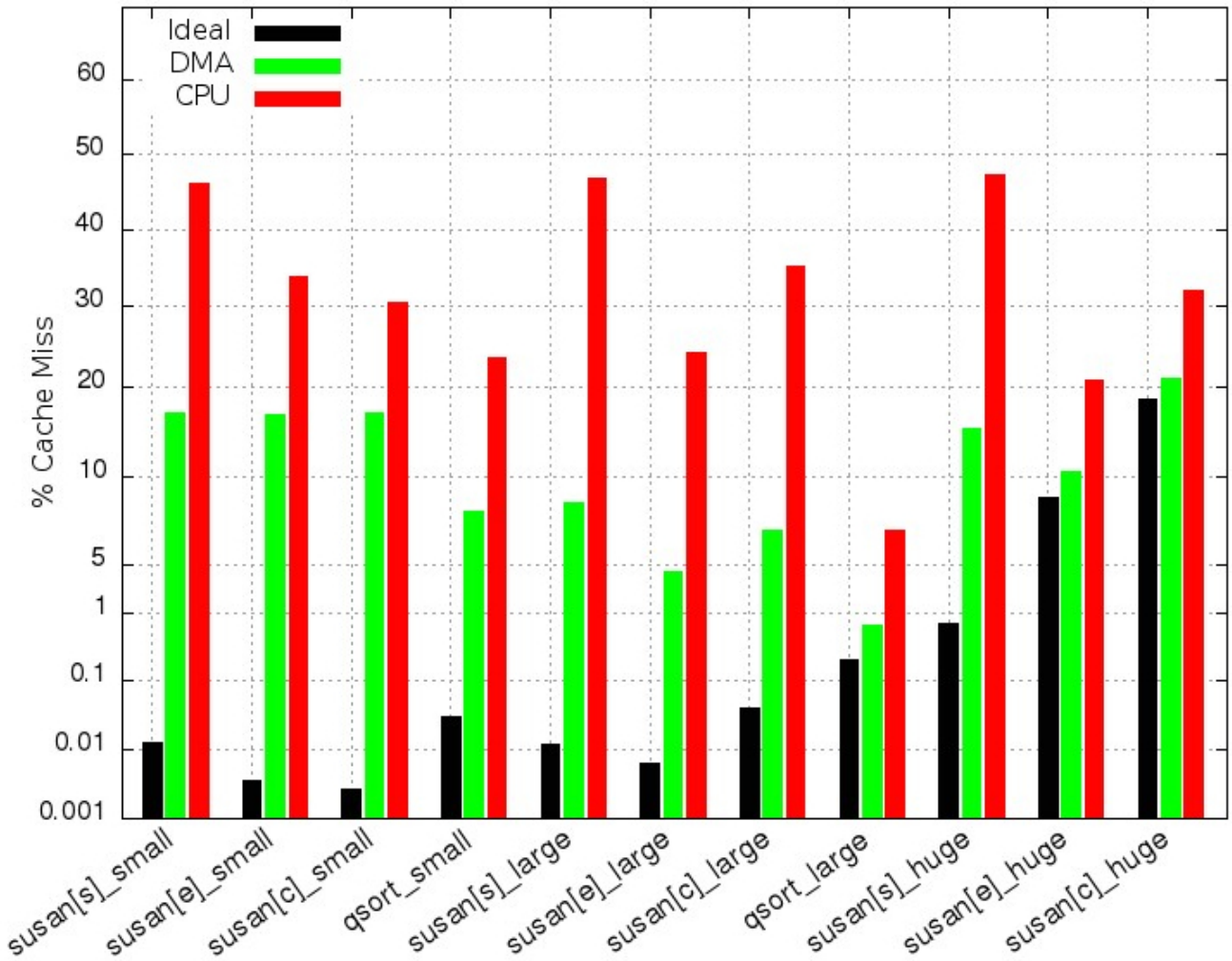
- *CH-HIGH* = chan0 e *CH-LOW* = chan1 , chan2, chan3



- Su buffer $>$ 64 KB: chan0: $\sim$ 1.2 GB/s, chan1-2-3: $\sim$ 0.65 GB/s

Cache Pollution: trasferimenti da 16064 Byte

- Per studiare il fenomeno della Cache Pollution sono stati utilizzati i test del benchmark MiBench



Conclusioni

- Le prestazioni del motore DMA sono migliori di quelle della CPU se i dati da trasferire non sono in memoria cache
- La velocità di trasferimento del motore DMA non è legata allo stato della memoria cache
- E' possibile definire varie politiche di scheduling per gestire, con un meccanismo di priorità, i canali DMA
- Nei trasferimenti DMA il fenomeno della Cache Pollution è circoscritto alle operazioni di setup
- L'estensione DMA-CFTU è 'indipendente' dall'hardware, con poche modifiche è utilizzabile su qualunque motore DMA che effettua operazioni di copia in modo asincrono

Sviluppi Futuri

- Sviluppo di politiche di scheduling più avanzate e integrazione in architetture Real-Time, per migliorare la predicibilità del sistema
- Memcpy tramite motore DMA
- Utilizzare il motore DMA per operazioni di XOR e MEMSET
- Processor DMA

Grazie per l'attenzione!