Università Degli Studi Di Roma Tor Vergata

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

Documentazione I/OAT-CFTU

Antonio Papa

Anno Accademico 2012-2013

Indice

Elenco delle figure

Strutturazione dei capitoli

Nel *Capitolo 1* verrà descritta la tecnologia I/OAT-CFTU, che permette al DMA di realizzare le operazioni di *copy_from_user()* e *copy_to_user()*.
Inoltre saranno introdotti:

- il meccanismo di profiling sul sistema per misurare la latenza delle copie in memoria (user-kernel space);
- alcuni micro-benchmark utilizzati per misurare le performance della CPU e del motore DMA.

Il *Capitolo 2* sarà dedicato all'analisi dei dati, al fine di quantificare i benefici della tecnologia I/OAT-CFTU.

Infine nel *Capitolo 3* verrà spiegato come:

- modificare il kernel al fine di utilizzare la tecnologia I/OAT-CFTU;
- utilizzare il meccanismo di profiling;
- effettuare delle misurazioni tramite micro-benchmark.

Capitolo 1

Architettura di Sistema

In questo capitolo verrà descritta la tecnologia I/OAT-CFTU e saranno presentati i meccanismi per l'analisi delle performance da me sviluppati.
E' possibile partizionare l'architettura su cui opera la l'I/OAT-CFTU in tre livelli (figura 1.1):

1. **Kernel Level:** le feature introdotte nel codice del kernel
2. **Module Level:** i driver virtuali usati come interfaccia Kernel-User
3. **User Level:** insieme di demoni atti alla generazione, raccolta ed analisi delle misure sulle performance del sistema.

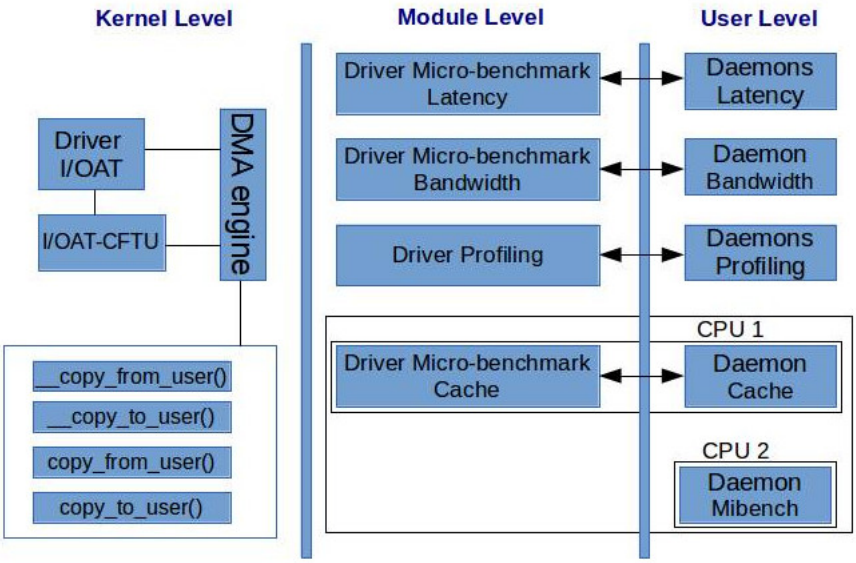

Figura 1.1: Architettura di Sistema I/OAT-CFTU

Questa suddivisione evidenzia il livello di iterazione, tra i componenti dell'architettura ed il sistema operativo sottostante, in termini di privilegi e risorse concessi.
Una classificazione più attenta alla logia di funzionamento dei componenti, che all'interazione con l'S.O. è la seguente:

1. **I/OAT-CFTU**
2. **Profiling Memory Operation**
3. **Micro-benchmark Latency**
4. **Micro-benchmark Bandwidth**
5. **Micro-benchmark Cache Pollution**

quest'ultimo partizionamento sarà preso come riferimento nel proseguo della trattazione.

1.1 I/OAT-CFTU

L'I/OAT-CFTU (ovvero Input/Output Acceleration Tecnology Copy From To User) è un insieme di tecnologie, che permettono ad un motore DMA di effettuare operazioni di copia che coinvolgono spazi di memoria diversi (Kernel-User). L'obiettivo è la riduzione dell'utilizzo della CPU durante le operazioni di *copy_from_user()* e *copy_to_user()*, al fine di aumentarne la predicibilità in ambito real-time. L'insieme di tecnologie introdotte nell'I/OAT-CFTU riguardano i seguenti aspetti:

1. **DCA & DMA Channe**l
2. **DMA Engine**
3. **Channel Policy**

1.1.1 DCA & DMA Channel

Come esposto precedentemente la predicibilità delle operazioni in un sistema Real-Time è fondamentale, quindi l'utilizzo di un componente impredicibile come la cache, mina l'affidabilità di suddetto sistema. Nel driver I/OAT è presente una funzionalità chiamata **DCA** (*Direct Cache Access*), che attiva un motore di pre-lettura (locato nella CPU) atto a caricare i dati nella cache, al fine di ridurre i miss. Per il motivo esposto sopra, questa funzionalità nell'I/OAT-CFTU è stata disattivata, portando ad un conseguente aumento dei tempi medi di latenza delle operazioni a beneficio di una minor varianza sui tempi delle suddette.
In alcuni motori DMA è possibile disattivare (*freezare*) i canali per un breve periodo, ma questa caratteristica hardware non è supportata dal driver base (I/OAT). Questa limitazione non consente di implementare politiche a priorità, in cui il bus del DMA è assegnato in modo esclusivo, per un determinato intervallo di una finestra temporale, ad uno o più canali. Per far fronte a questo problema l'I/OAT-CFTU implementa un'interfaccia atta alla sospensione/risveglio dei canali DMA.

1.1.2 DMA Engine

Il DMA Engine è un sottosistema, che fornisce un insieme di interfacce hw-neutral (indipendenti dall'architettura hardware) che possono essere usate su un qualunque DMA

con tecnologia I/OAT. Ogni motore DMA può supportare diversi tipi di transazione (come mostrato nel codice sottostante, locato in: include/linux/dmaengine.h), che permettono di differenziare e specializzare le operazioni eseguite.

```
enum dma_transaction_type {
  DMA_MEMCPY,
  DMA_XOR,
  DMA_PQ,
  DMA_XOR_VAL,
  DMA_PQ_VAL,
  DMA_MEMSET,
  DMA_INTERRUPT,
  DMA_SG,
  DMA_PRIVATE,
  DMA_ASYNC_TX,
  DMA_SLAVE,
  DMA_CYCLIC,
  DMA_INTERLEAVE,
/* last transaction type for creation of the capabilities mask */
  DMA_TX_TYPE_END,
};
```

Come è facile notare, la tipologia di nostro interesse è classificata con **DMA_MEMCPY**, questa include tutti i DMA che supportano operazioni di tipo *Memory to Memory*. Quindi al fine di supportare le operazioni di *copy_from_user()* e *copy_to_user()* è stato necessario estendere l'interfaccia DMA Engine, aggiungendo funzioni che consentissero la copia tra due regioni di memoria distinta locate in due diversi space (user e kernel).

1.1.3 Channel Policy

Il DMA Engine come spiegato sopra fornisce un sottosistema ricco di funzionalità, ma non implementa nessuna politica per la gestione dei canali DMA. L'I/OAT-CFTU, per sopperire a questa mancanza, introduce un insieme di politiche per la gestione dei canali DMA, politiche che possono essere suddivise in due grandi famiglie:

- **Politiche Statiche**
- **Politiche Dinamiche**

Le Politiche Statiche vengono utilizzate per integrare le funzionalità del DMA direttamente nel codice delle funzioni: *__copy_from_user()*, *__copy_to_user()*, *copy_from_user()* e *copy_to_user()*. Grazie a questa famiglia il S.O. utilizza il DMA (anche nella fase di boot del kernel) per effettuare le copie tra memory-space, ma il prezzo da pagare per tale supporto è appunto la staticità. Infatti suddette politiche vengono selezionate in fase di config del kernel, quindi non è possibile cambiare politica senza ricompilare. L'I/OAT-CFTU implementa 2 politiche statiche:

1. **Exclusive Channel**
2. **Shared Channel**

Nella prima politica ad ogni CPU del sistema è associato in modo esclusivo un canale DMA, questo permette di utilizzare sui canali DMA il bilanciamento del carico effettuato sulle CPU. Nella seconda politica i canali DMA sono condivisi quindi non ci sono vincoli circa il loro utilizzo.
Le Politiche Dinamiche invece possono essere cambiate 'a caldo', ossia anche a sistema avviato. Infatti in questo caso le funzioni di copia del kernel non vengono modificate, ma ogni politica mette a disposizione delle API per le operazioni di copia tramite DMA. Ogni politica dinamica è inizializzata-lanciata e rimossa tramite le funzione di *start_ <name_ policy>()* e *stop_ <name_ policy>()*. L'I/OAT-CFTU implementa 3 politiche di tipo dinamico:

1. **Exclusive Channel**
2. **Shared Channel**
3. **Priority Channel**

Le prime due sono identiche a quelle statiche, eccezion fatta per il loro utilizzo 'a caldo'. La terza implementa una gestione delle risorse DMA basata sulla priorità dei canali, ossia è possibile dedicare il bus DMA per un certo quanto temporale (il quanto è da intendersi come una porzione di una finestra temporale), ad uno o più canali.

1.2 Profiling Memory Operation

Il Profiling è un'alternativa al benchmarking, che è spesso più efficace, in quanto permette di ottenere misurazioni più precise per i diversi componenti del sistema che si intende analizzare. Componenti che nel nostro specifico caso sono le operazioni sulla memoria: *__copy_from_user()*, *__copy_to_user()*, *copy_from_user()* e *copy_to_user()*. Il tool per il profiling permette, tramite l'uso di buffer circolari per-cpu, di ridurre al minimo i tempi di setup e raccolta delle misure. Il kernel nella fase di boot alloca suddetti buffer (uno per ogni operazione da misurare), che saranno usati per raccogliere i dati sui tempi di latenza delle funzioni di interesse. I buffer circolari sono implementati con una politica di tipo *Wine*, quindi i dati non saranno sovrascritti a buffer pieno. Come è facile intuire, le operazioni che devono essere misurate scriveranno dati nei buffer, mentre il ruolo di lettore sarà ricoperto da un driver virtuale, che tramite device file fornisce i dati allo spazio utente. Nello spazio utente risiederanno dei demoni che periodicamente leggeranno suddetti device file per ottenere le misurazioni di loro interesse. Così grazie al tool Profiling Memory Operation è possibile esportare in user space le misurazioni effettuate in kernel space. Questo tipo di profiling può essere eseguito su:

- sistemi che operano in maniera standard, ossia che eseguono le operazioni sulla memoria tramite CPU. In questo caso si avrà come risultato il trace dei tempi di copia della CPU.
- Sistemi che utilizzano una delle Politiche Statiche precedentemente introdotte. In questo caso si avrà un trace sui tempi di copia del DMA.

1.3 Micro-benchmark Latency & Bandwidth

Il tool per il profiling sopra introdotto ha un limite, quello di inserire nelle misure i tempi di pin/unpin delle pagine user. Infatti una copia tramite DMA coinvolge una o più pagine di memoria dello spazio utente, che necessitano di essere pinnate per evitare un loro eventuale swapping su disco. Ma in un sistema embedded tutto ciò è superfluo, quindi bisogna eliminare suddetti tempi dalle misurazioni. Una soluzione possibile consiste in un controllo preventivo della regione di memoria user, allo scopo di effettuare il pin prima della misurazione e l'unpin dopo. Quanto appena asserito non è realizzabile nel meccanismo di profiling, ma tramite micro-benchmark dove l'input è controllato.
Quindi si è scelto di implementare due tipologie di micro-benchmark:

1. **Micro-benchmark Latency**
2. **Micro-benchmark Bandwidth**

Il primo micro-benchamrk è utilizzato per misurare i tempi di latenza delle operazioni di copia, che possono essere di tipo:

- *standard*: *copy_from_user()*, *copy_to_user()* eseguite sulla CPU
- *DMA*: *copy_from_user*, *copy_to_user* implementate nelle politiche dinamiche *Exclusive* e *Shared*.

Purtroppo il micro-benchmark sulla latenza ha due limiti: le operazioni misurate sono eseguite su un solo canale DMA, quindi non si sfrutta completamente la larghezza di banda del Bus DMA; e non è adatto per effettuare test sulla politica dinamica *Priority Channel*. Per far fronte a queste limitazioni è stato implementato un secondo micro-benchamrk, che ha lo scopo di misurare la Bandwidth parallelizzando le operazioni di copia (che coinvolgono due regioni di memoria con una certa size) su tutte le risorse del sistema (CPU o Canali DMA in base all'operazione misurata). Quindi il secondo micro-benchmark può anche essere utilizzato per testare le operazioni fornite dalla politica dinamica *Priority Channel*. Infatti misurando la velocità di banda dei diversi canali DMA è possibile comprendere il comportamento del/dei canale/i a priorità alta rispetto ai restanti.

1.4 Micro-benchmark Cache Pollution

Quest'ultimo micro-benchmark è utilizzato per misurare le performance relative ai cache miss ed execution time di applicazioni che sono eseguite in parallelo ad operazioni sulla memoria (*copy_to/from_user()*) effettuate tramite CPU o DMA. Le applicazioni da misurare saranno eseguite su una CPU diversa da quella utilizzata per le operazioni di memoria, le due CPU devono però condividere lo stesso socket per poter usare la stessa *cache L2*. Questo ci permette di misurare l'inquinamento della cache (*Cache pollution*) generato tramite operazioni sulla memoria dal DMA e dalla CPU e quindi gli effetti che il *Cache Polluction* ha sull'*execution time* di determinate applicazioni.

Il micro-benchmark sarà quindi composto da due entità in esecuzione su CPU diverse (es: CPU 1 e CPU2):

- *Unit Cache Polluction*: in esecuzione sulla CPU 0, effettua operazioni sulla memoria via DMA o CPU
- *Unit Application*: in esecuzione sulla CPU 1, esegue alcuni test del benchmark Mibench (categoria automotive) e ne misura le performance.

Capitolo 2

Analisi dei dati

Si passerà ora a presentare e discutere i risultati ottenuti tramite i micro-benchmark precedentemente descritti.
Al fine di considerare anche lo stato di carico del sistema operativo, i test qui presentati sono stati effettuati in due scenari diversi:

- **System light-load:** il sistema operativo durante il test non deve gestire processi che causano un heavy-load.
- **System heavy-load:** il sistema operativo durante il test deve gestire processi che causano un heavy-load.

Infine descriviamo brevemente l'architettura hardware-software utilizzata per i test:

- **Server Familiy:** Apple Xserve
- **CPU 1:** Intel(R) Xeon(R) CPU 5150 @ 2.66GHz 64 bit
- **CPU 2:** Intel(R) Xeon(R) CPU 5150 @ 2.66GHz 64 bit
- **RAM:** 4 GB
- **MCH:** 5000X Chipset Memory Controller Hub
- **DMA:** 5000 Series Chipset DMA Engine 64 bit

2.1 Latency Performance

Questo paragrafo sarà dedicato all'analisi dei tempi di latenza delle copie in memoria che usano la CPU o il DMA. Si calcolerà la latenza delle suddette operazioni:

- **copy_from_user**
- **copy_from_user (no cache)**
- **copy_to_user**
- **copy_to_user (no cache)**

Al fine di mostrare le prestazioni della CPU con e senza cache saranno mostrati anche i tempi di latenza delle *copy_from_user* e *copy_to_user* (no cache). Queste due operazioni si distinguono dalle altre, in quanto prima della copia in memoria viene azzerata la loro porzione di cache. Ogni operazione sarà eseguita in tre contesti hardware-software diversi:

- **CPU:** l'operazione di copia è eseguita completamente sulla CPU
- **DMA Excl:** l'operazione di copia è eseguita dal motore DMA che opera con politica Exclusive Channel
- **DMA Sh:** l'operazione di copia è eseguita dal motore DMA che opera con politica Shared Channel

Per dare una panoramica esaustiva si è deciso di effettuare le misure su diverse grandezze di blocchi di memoria, l'intorno preso come riferimento è il seguente $[1 - 2^{25}]$ byte.

2.1.1 System light-load

Lo scenario in considerazione è un sistema light-load in cui sarà eseguito il micro-benchmark realtivo alla latenza. Per ogni diversa size del blocco di memoria sono state effettuate 300 misurazioni, il valore graficato della latenza è la media di queste misurazioni.

La Figura 2.1 mostra il tempo di latenza del sistema nell'eseguire l'operazione di copy_from_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

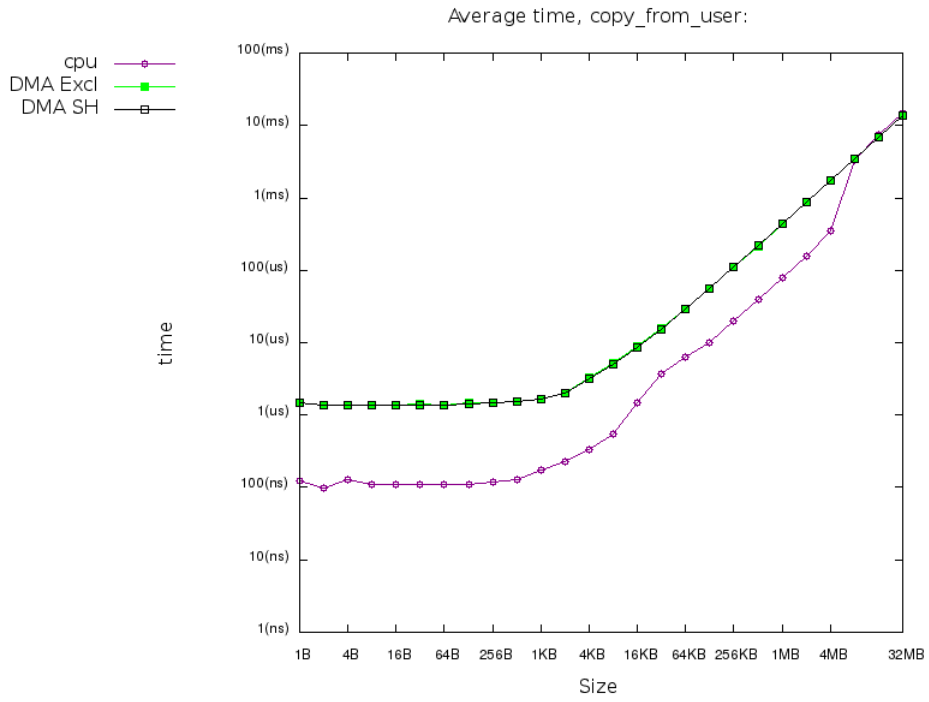

Figura 2.1: [System light-load] Tempo di latenza copy_from_user()

La Figura 2.2 mostra il tempo di latenza del sistema nell'eseguire l'operazione di copy_to_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

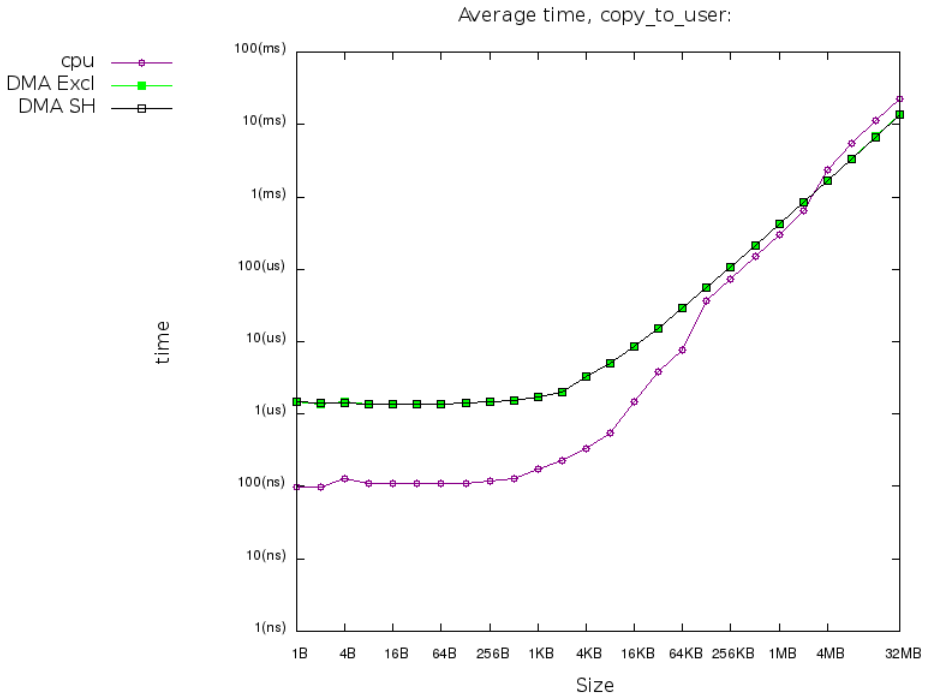

Figura 2.2: [System light-load] Tempo di latenza copy_to_user()

La Figura 2.3 mostra il tempo di latenza del sistema nell'eseguire l'operazione di copy_from_user() (no cache) al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

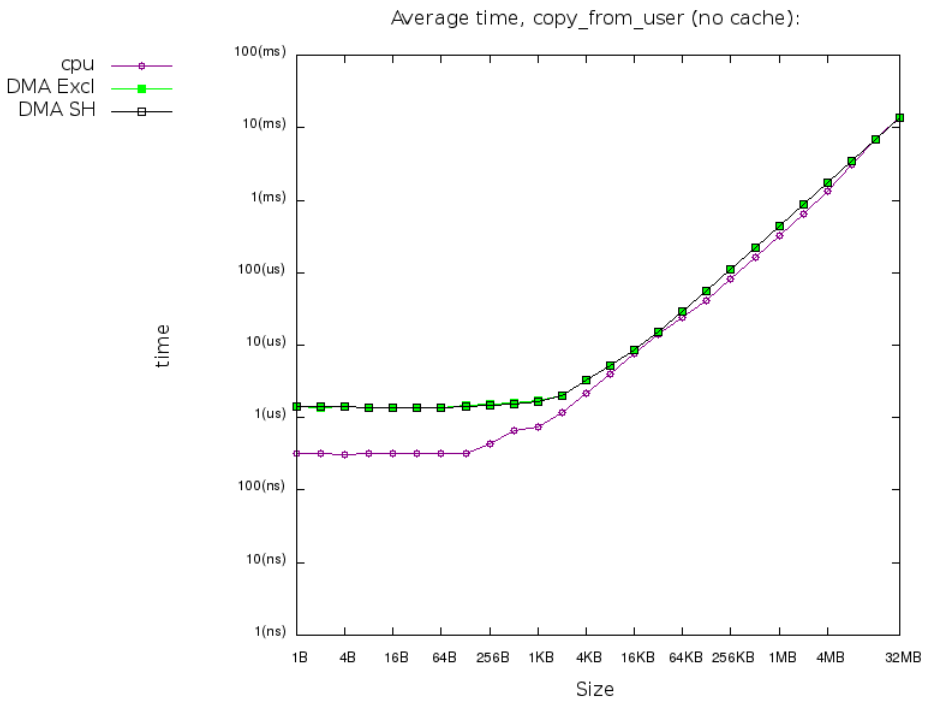

Figura 2.3: [System light-load] Tempo di latenza copy_from_user() (no cache)

La Figura 2.4 mostra il tempo di latenza del sistema nell'eseguire l'operazione di copy_to_user() (no cache) al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

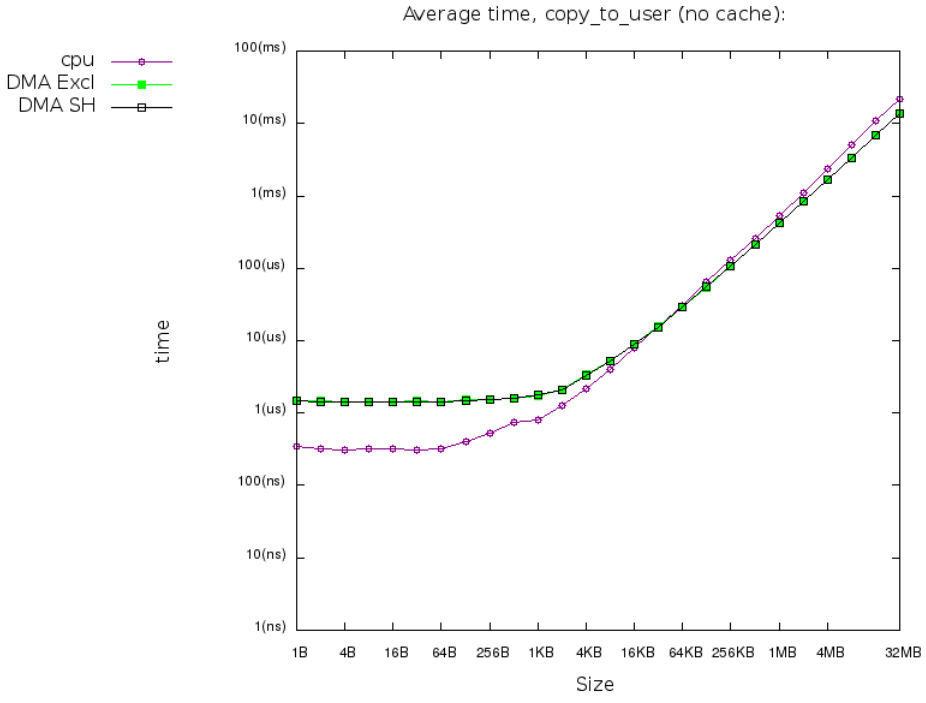

Figura 2.4: [System light-load] Tempo di latenza copy_to_user() (no cache)

2.1.2 System heavy-load

Lo scenario in considerazione è un sistema heavy-load in cui sarà eseguito il micro-benchmark realtivo alla latenza. Per ogni diversa size del blocco di memoria sono state effettuate 300 misurazioni, il valore graficato della latenza è la media di queste misurazioni.

La Figura 2.5 mostra il tempo di latenza del sistema nell'eseguire l'operazione di copy_from_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

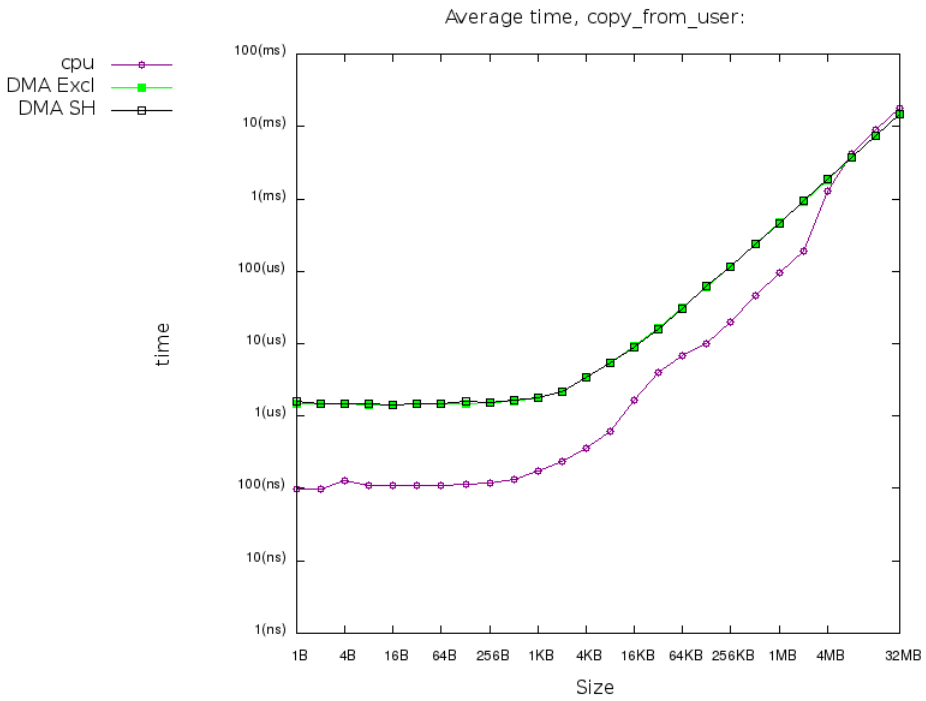

Figura 2.5: [System heavy-load] Tempo di latenza copy_from_user()

La Figura 2.6 mostra il tempo di latenza del sistema nell'eseguire l'operazione di copy_to_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

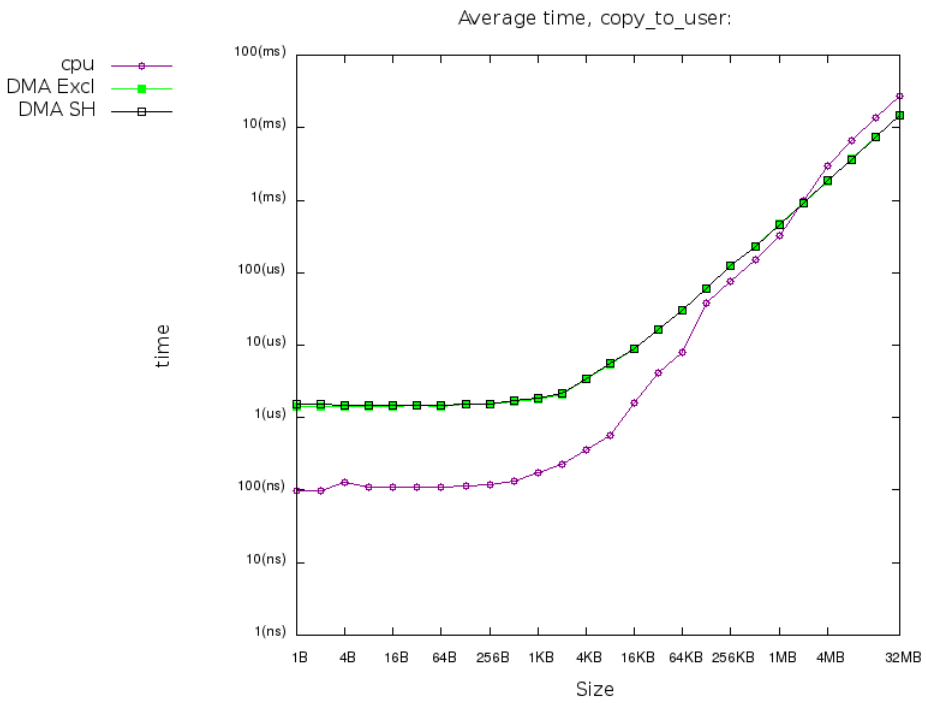

Figura 2.6: [System heavy-load] Tempo di latenza copy_to_user()

La Figura 2.7 mostra il tempo di latenza del sistema nell'eseguire l'operazione di copy_from_user() (no cache) al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

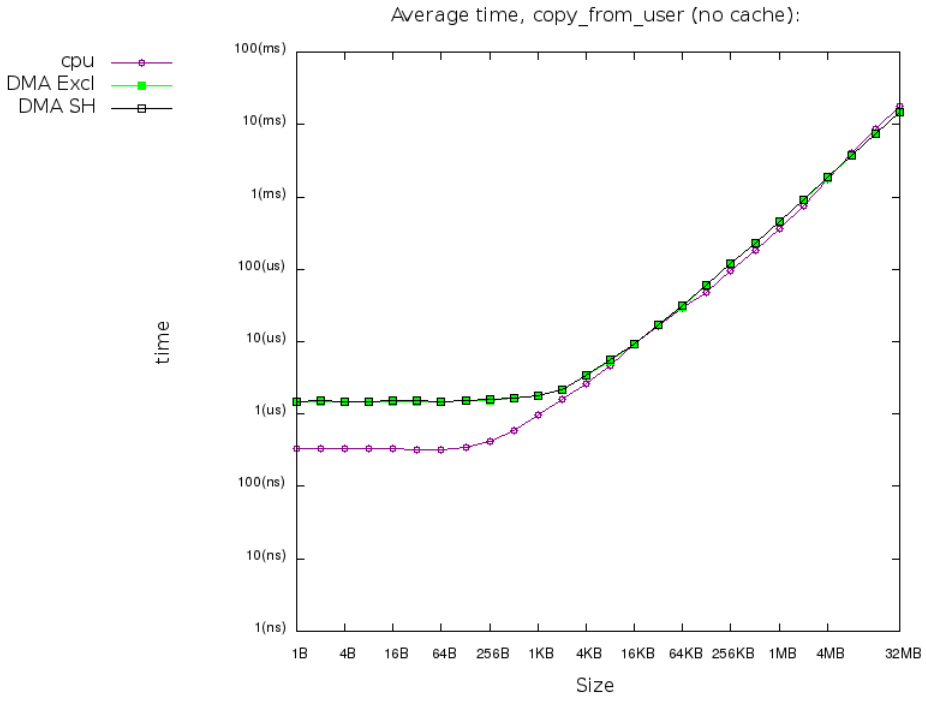

Figura 2.7: [System heavy-load] Tempo di latenza copy_from_user() (no cache)

La Figura 2.8 mostra il tempo di latenza del sistema nell'eseguire l'operazione di co-

py_to_user() (no cache) al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

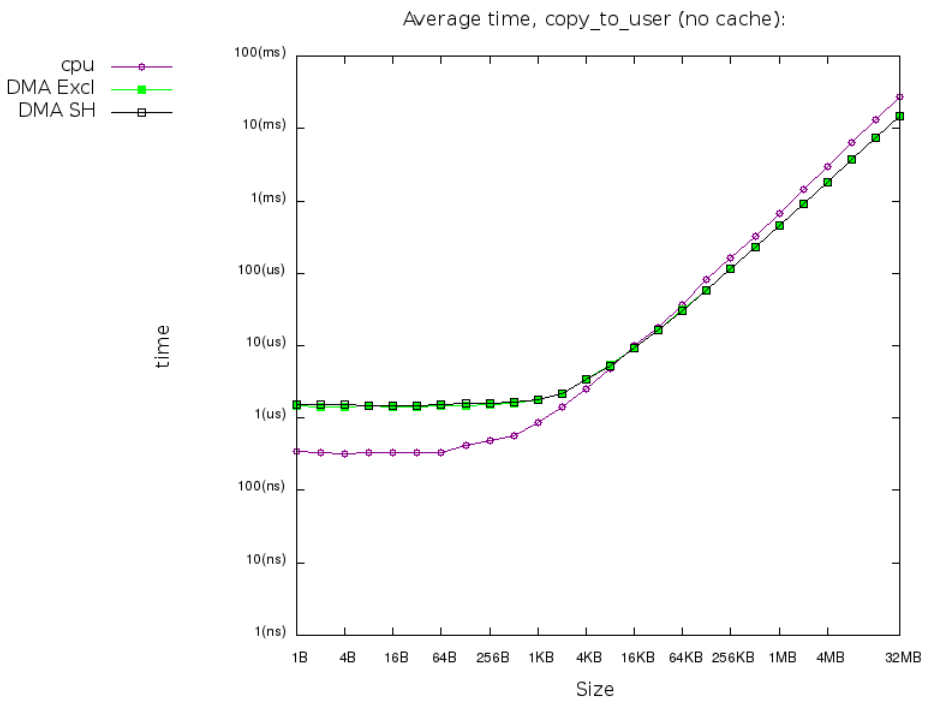

Figura 2.8: [System heavy-load] Tempo di latenza copy_to_user() (no cache)

2.2 Bandwidth Performance

Questo paragrafo sarà dedicato all'analisi della bandwidth della CPU e del DMA mentre effettuano operazioni di copia. Questa analisi permette di verificare quali siano le reali performance del DMA, infatti i test del paragrafo precedente coinvolgevano un solo canale del motore DMA. In questa casistica invece ogni canale, nello stesso intorno temporale, deve espletare un certo numero di operazioni. Si calcolerà la velocità di trasmissione su 3 tipologie di operazioni:

- **copy_from_user**
- **copy_to_user**
- **copy_from_user + copy_to_user**

Ogni operazione sarà eseguita in 4 contesti hardware-software diversi:

1. **CPU:** l'operazione di copia è eseguita completamente sulla CPU
2. **DMA Excl:** l'operazione di copia è eseguita dal motore DMA che opera con politica Exclusive Channel
3. **DMA Prio:** l'operazione di copia è eseguita dal motore DMA che opera con politica Priority Channel (un canale a priorità alta che per un quanto di tempo ha l'uso esclusivo del DMA)

4. **DMA Sh:** l'operazione di copia è eseguita dal motore DMA che opera con politica Shared Channel

E' bene specificare che nei test che seguiranno si considerà la Bandwidth totale, ossia la somma delle velocità di trasmissione su tutte le CPU nel contesto(1) o su tutti i canali DMA nei contesti$(2, 3, 4)$. Inoltre nel contesto **DMA Prio** il quanto di tempo del canale a priorità alta è stato settato a $30usec$ su una finestra temporale di $90usec$. Per dare una panoramica esaustiva si è deciso di effettuare le misure su diverse grandezze di blocchi di memoria, l'intorno preso come riferimento è il seguente $[2^{10} - 2^{25}]$ byte.

2.2.1 System light-load

Lo scenario in considerazione è un sistema light-load in cui sarà eseguito il micro-benchmark realtivo alla bandwidth. Per ogni diversa size del blocco di memoria sono state effettuate 300 operazioni dello stesso tipo.

La Figura 2.9 mostra la velocità di trasmissione del sistema nell'eseguire l'operazione di copy_from_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

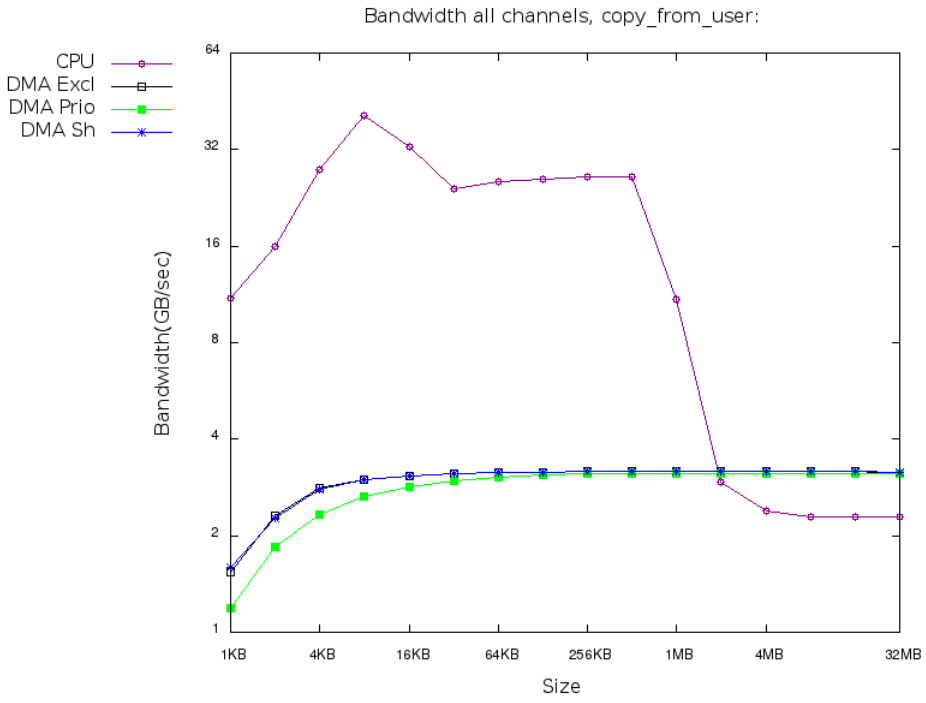

Figura 2.9: [System light-load] Bandwidth copy_from_user()

La Figura 2.10 mostra la velocità di trasmissione del sistema nell'eseguire l'operazione di copy_to_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

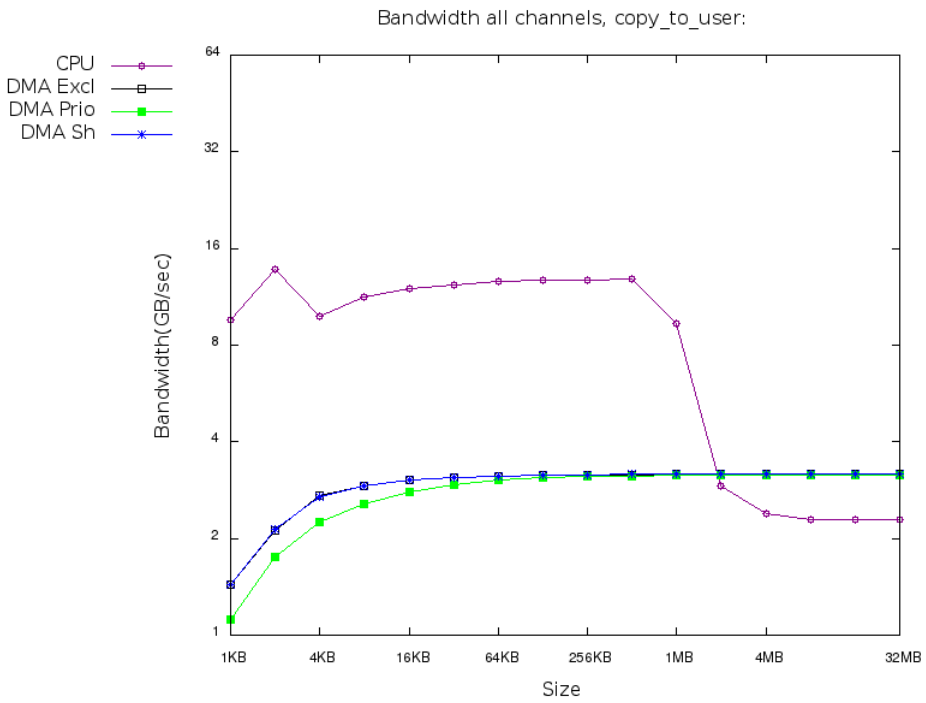

Figura 2.10: [System light-load] Bandwidth copy_ to_ user()

La Figura 2.11 mostra la velocità di trasmissione del sistema nell'eseguire l'operazione di copy_from_user() + copy_to_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

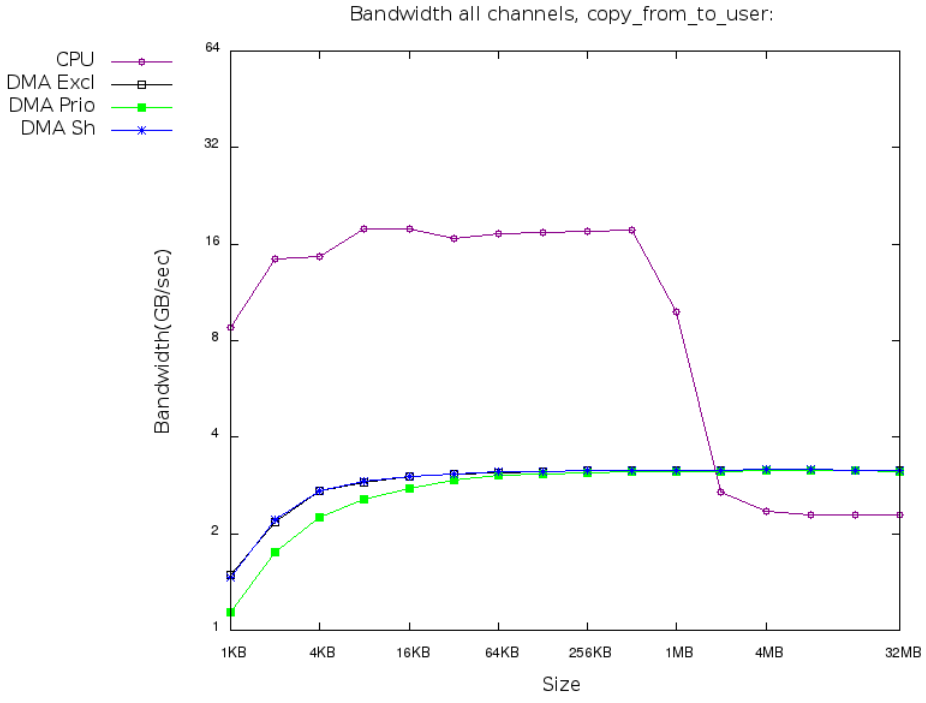

Figura 2.11: [System light-load] Bandwidth copy_from_ user() + copy_ to_ user()

2.2.2 System heavy-load

Lo scenario in considerazione è un sistema heavy-load in cui sarà eseguito il micro-benchmark realtivo alla bandwidth. Per ogni diversa size del blocco di memoria sono state effettuate 300 operazioni dello stesso tipo.

La Figura 2.12 mostra la velocità di trasmissione del sistema nell'eseguire l'operazione di copy_from_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

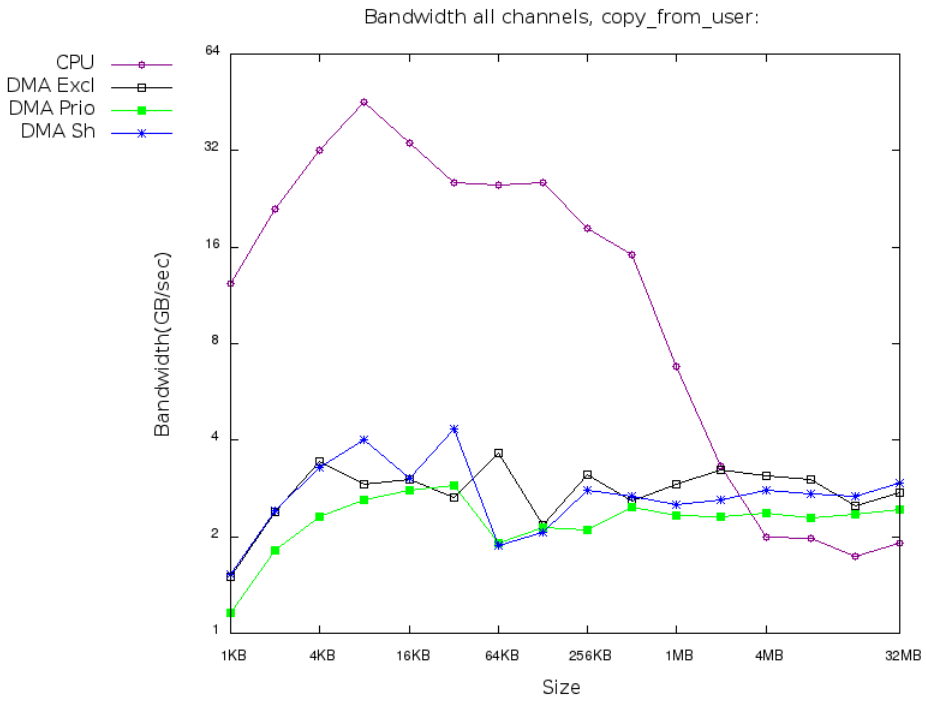

Figura 2.12: [System heavy-load] Bandwidth della copy_from_user()

La Figura 2.13 mostra la velocità di trasmissione del sistema nell'eseguire l'operazione di copy_to_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

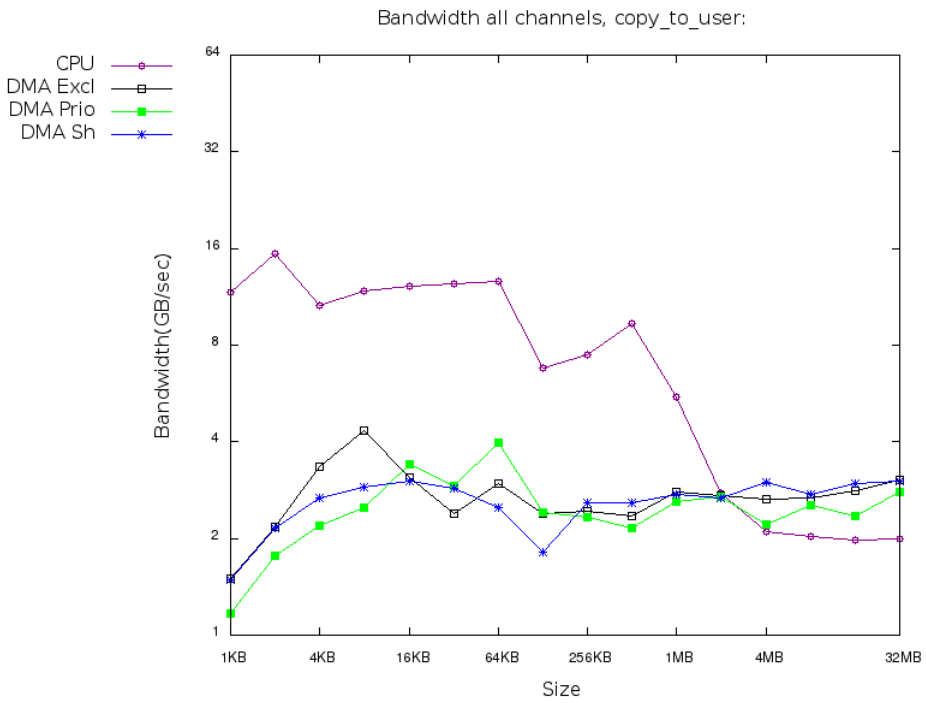

Figura 2.13: [System heavy-load] Bandwidth della copy_to_user()

La Figura 2.14 mostra la velocità di trasmissione del sistema nell'eseguire l'operazione di copy_from_user() + copy_to_user() al variare della grandezza dei blocchi di memoria. Il test è stato ripetuto per i diversi contesti hardware-software precedentemente elencati.

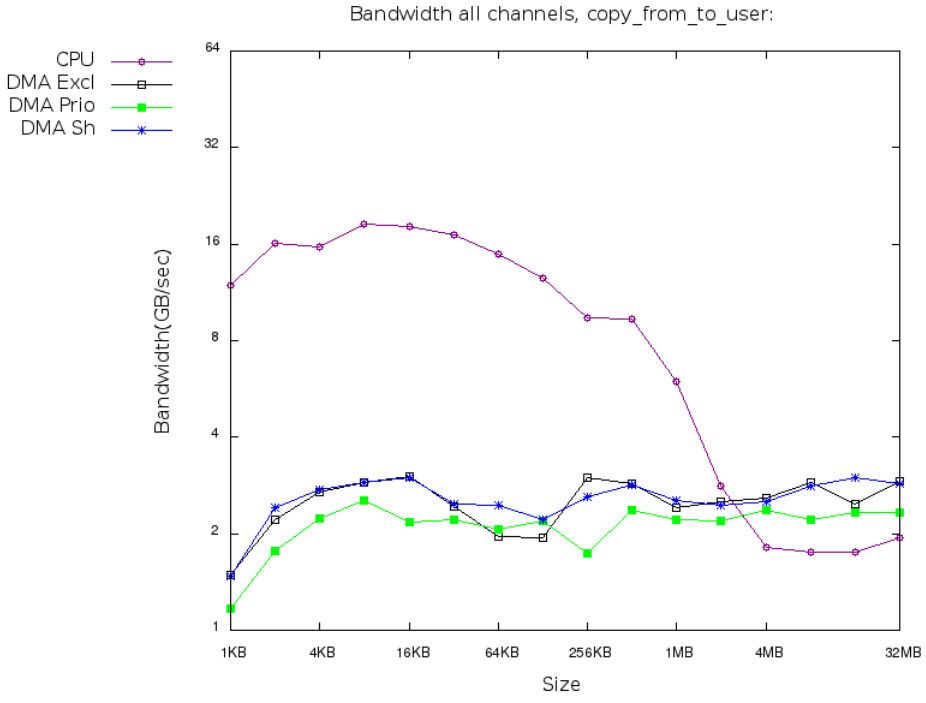

Figura 2.14: [System heavy-load] Bandwidth della copy_from_user() + copy_to_user()

2.3 Priority Channel Performance

Infine analizziamo la bandwidth sui singoli canali del DMA che opera con politica Priority Channel. Si calcolerà la velocità di trasmissione su 3 tipologie di operazioni:

- **copy_from_user**
- **copy_to_user**
- **copy_from_user + copy_to_user**

Il quanto di tempo del canale a priorità alta è stato settato a 30 μs su una finestra temporale di 90 μs. Per dare una panoramica esaustiva si è deciso di effettuare le misure su diverse grandezze di blocchi di memoria, l'intorno preso come riferimento è il seguente $[2^{10} - 2^{25}]$ byte.

2.3.1 System light-load

Lo scenario in considerazione è un sistema light-load in cui sarà eseguito il micro-benchmark realtivo alla bandwidth. Per ogni diversa size del blocco di memoria sono state effettuate 300 operazioni dello stesso tipo.

La Figura 2.15 mostra la velocità di trasmissione dei canali DMA nell'eseguire l'operazione di copy_from_user() al variare della grandezza dei blocchi di memoria.

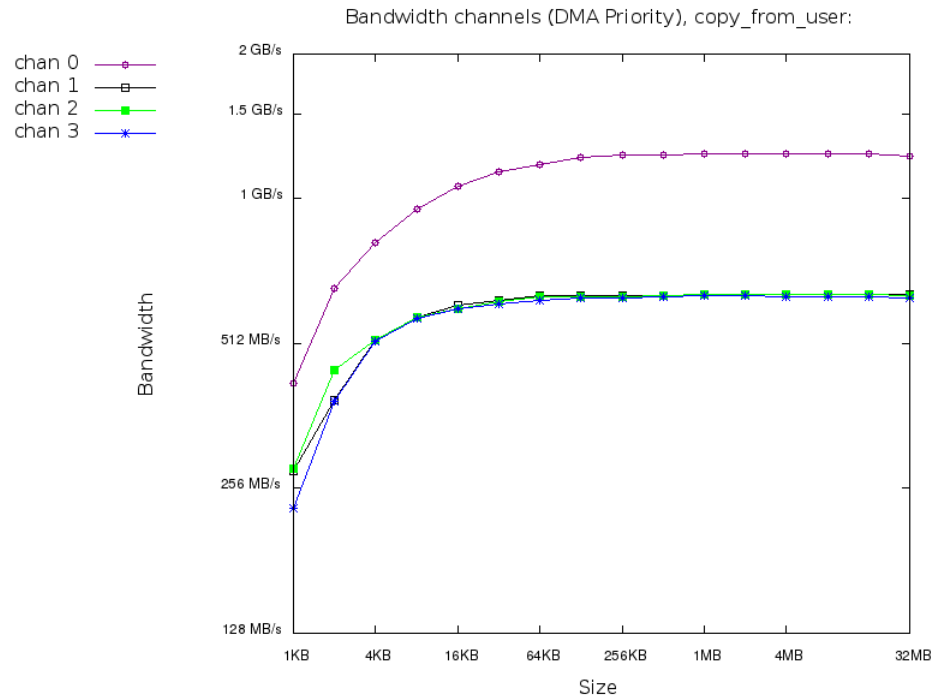

Figura 2.15: [System light-load] Bandwidth Priority Channel copy_from_user()

La Figura 2.16 mostra la velocità di trasmissione dei canali DMA nell'eseguire l'operazione di copy_to_user() al variare della grandezza dei blocchi di memoria.

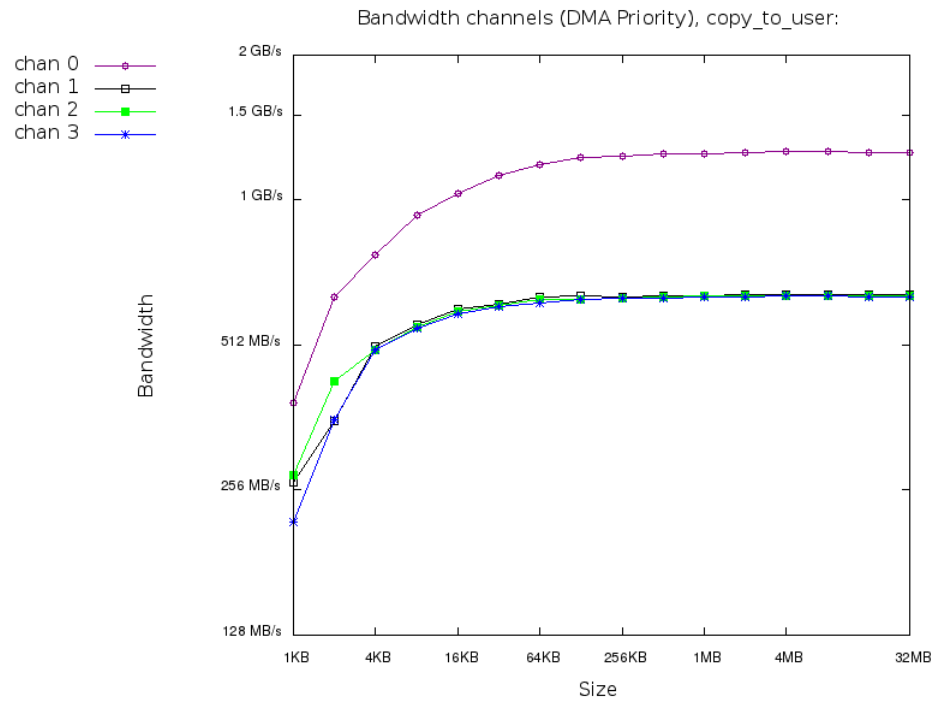

Figura 2.16: [System light-load] Bandwidth Priority Channel copy_to_user()

La Figura 2.17 mostra la velocità di trasmissione dei canali DMA nell'eseguire l'operazione di copy_from_user() + copy_to_user() al variare della grandezza dei blocchi di memoria.

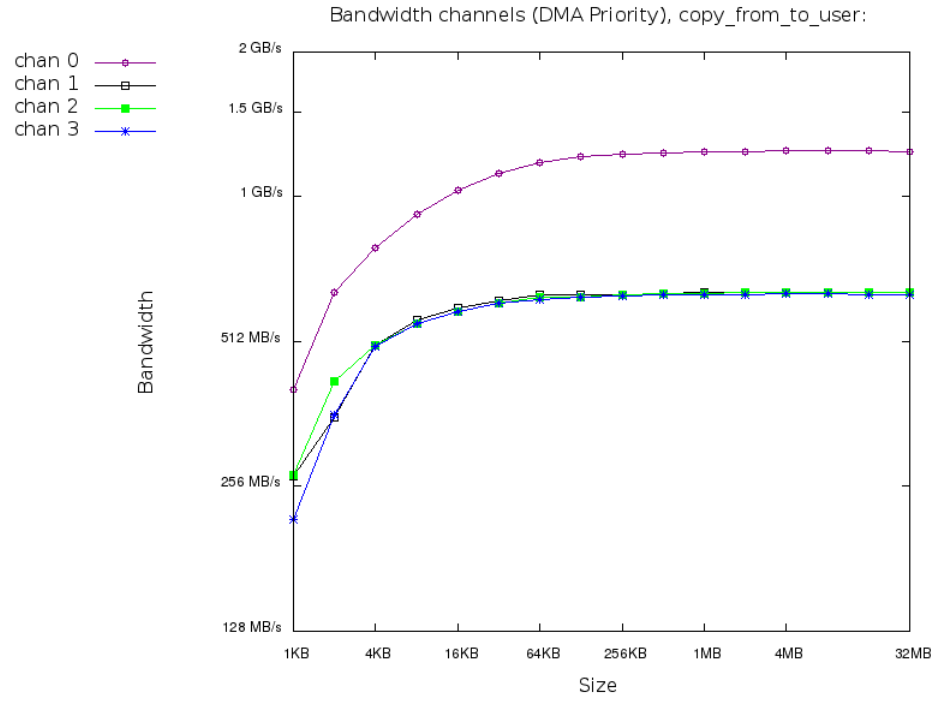

Figura 2.17: [System light-load] Bandwidth Priority Channel copy_from_user() + copy_to_user()

2.3.2 System heavy-load

Lo scenario in considerazione è un sistema heavy-load in cui sarà eseguito il micro-benchmark realtivo alla bandwidth. Per ogni diversa size del blocco di memoria sono state effettuate 300 operazioni dello stesso tipo.

La Figura 2.18 mostra la velocità di trasmissione dei canali DMA nell'eseguire l'operazione di copy_from_user() al variare della grandezza dei blocchi di memoria.

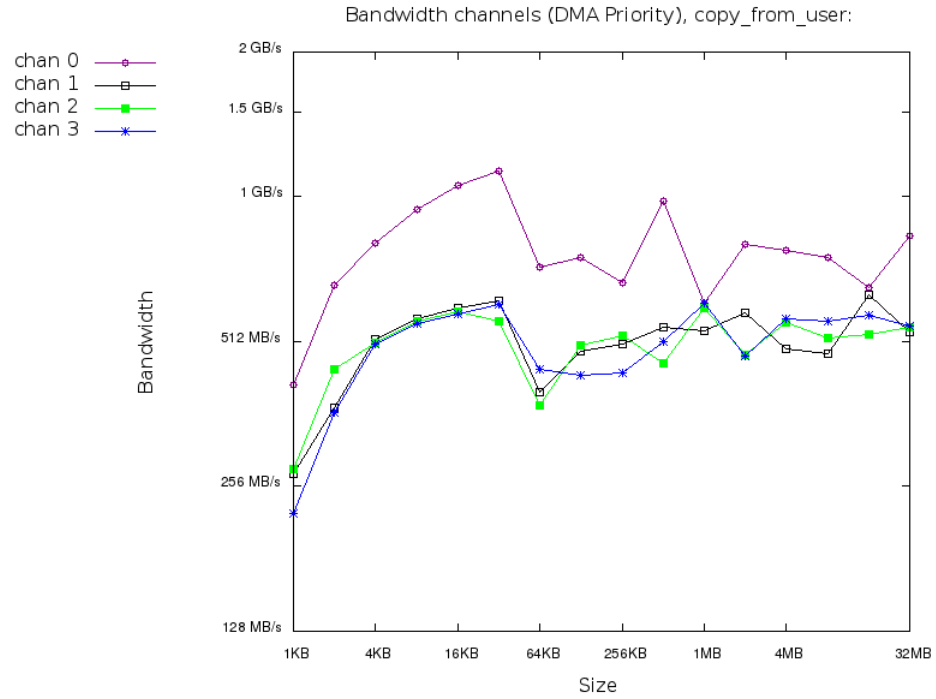

Figura 2.18: [System heavy-load] Bandwidth Priority Channel copy_from_user()

La Figura 2.19 mostra la velocità di trasmissione dei canali DMA nell'eseguire l'operazione di copy_to_user() al variare della grandezza dei blocchi di memoria.

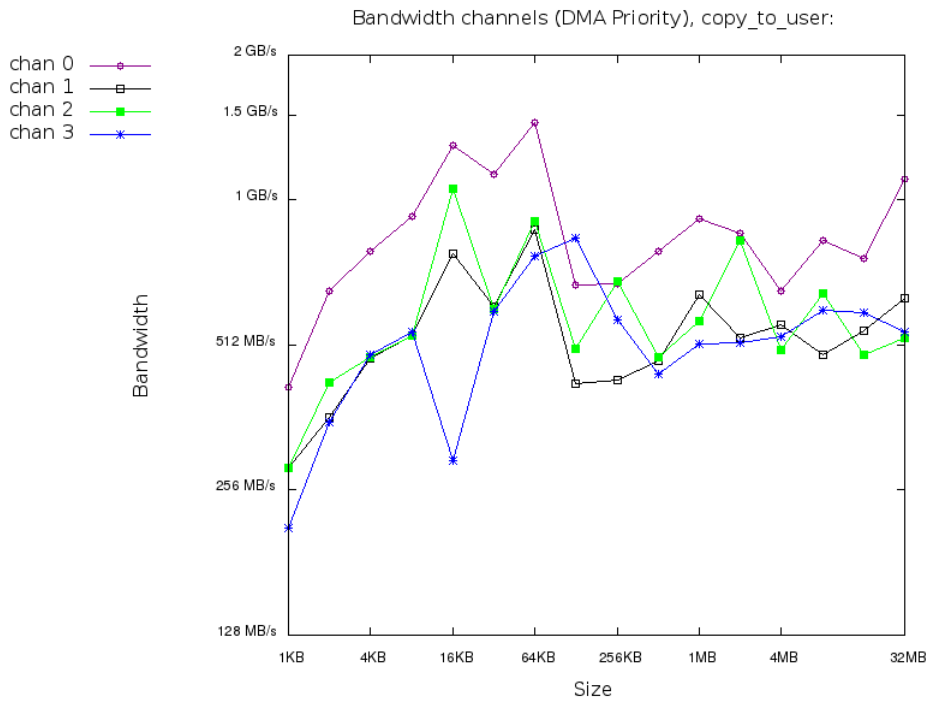

Figura 2.19: [System heavy-load] Bandwidth Priority Channel copy_to_user()

La Figura 2.20 mostra la velocità di trasmissione dei canali DMA nell'eseguire l'operazione di copy_from_user() + copy_to_user() al variare della grandezza dei blocchi di memoria.

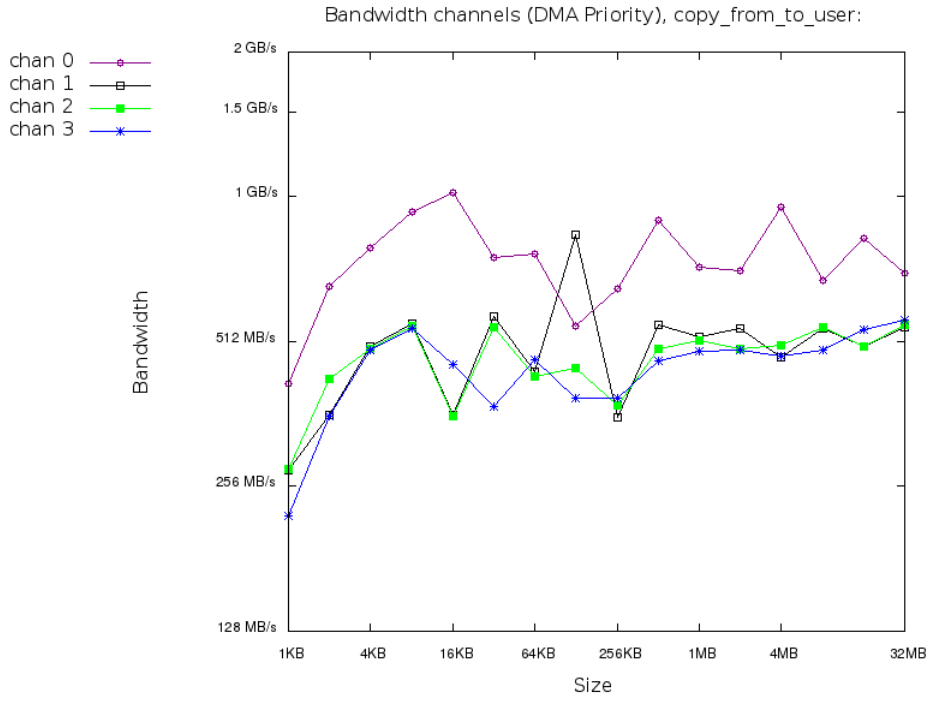

Figura 2.20: [System heavy-load] Bandwidth Priority Channel copy_from_user() + copy_to_user()

2.4 System Trace

In questo paragrafo si studierà come il Sistema Operativo utilizza le operazioni sulla memoria (User-Kernel), attraverso i trace generati dal tool di profiling sopra descritto. Infatti analizzando i trace è possibile calcolare, per un dato intorno temporale, quante volte una determinata operazione è stata chiamata dal Kernel e partizionare suddette invocazioni in base alla grandezza del blocco di memoria su cui operano.
Le operazioni studiate saranno le suddette:

- **copy_from_user**
- **copy_to_user**
- **__copy_from_user**
- **__copy_to_user**

La Figura 2.21 mostra il numero di invocazioni della funzione *copy_from_user()* in una finestra temporale di 100 secondi:

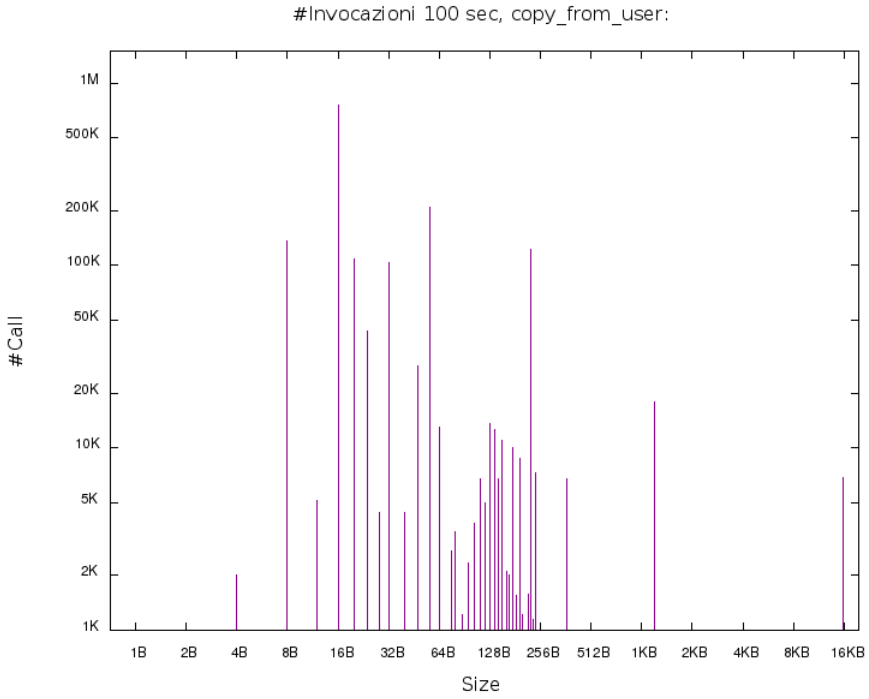

Figura 2.21: Profiling copy_from_user()

La Figura 2.22 mostra il numero di invocazioni della funzione *copy_to_user()* in una finestra temporale di 100 secondi:

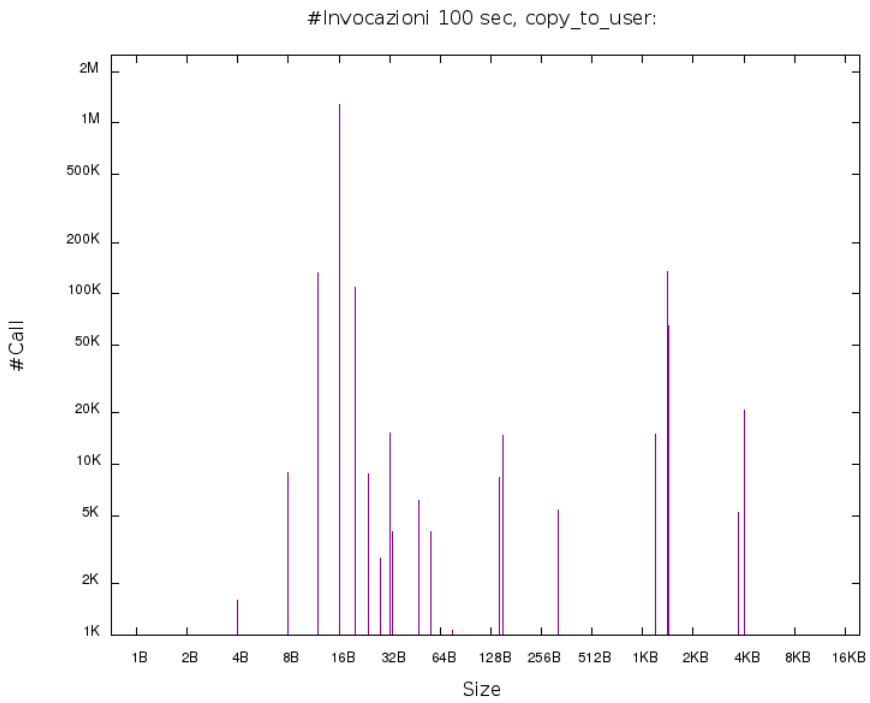

Figura 2.22: Profiling copy_to_user()

La Figura 2.23 mostra il numero di invocazioni della funzione *__copy_from_user()* in una finestra temporale di 100 secondi:

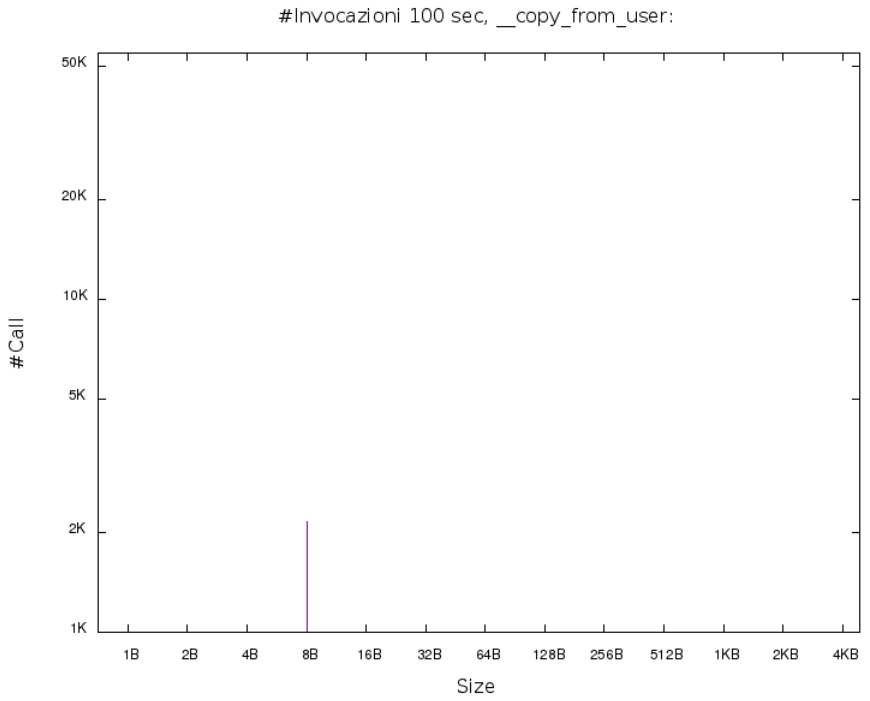

Figura 2.23: Profiling __copy_from_user()

La Figura 2.24 mostra il numero di invocazioni della funzione *__copy_to_user()* in una finestra temporale di 100 secondi:

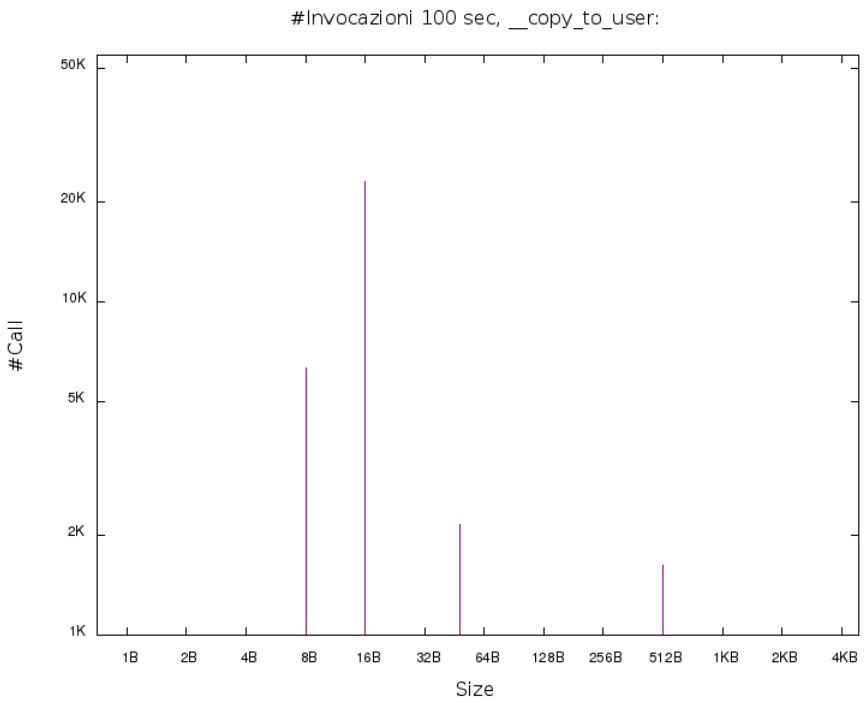

Figura 2.24: Profiling __copy_to_user()

Infine mostriamo la somma del numero di invocazioni delle funzioni *copy_from_user()*,*__copy_from_user()* in Figura 2.25 e quelle delle funzioni *copy_to_user()*, *__copy_to_user()* in Figura 2.26.

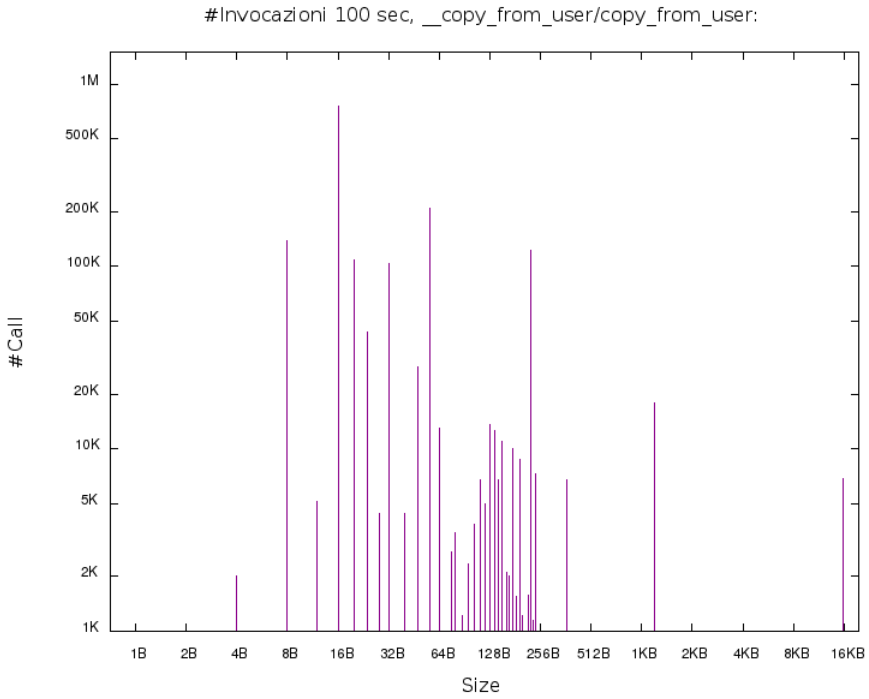

Figura 2.25: Profiling copy_from_user() e __copy_from_user()

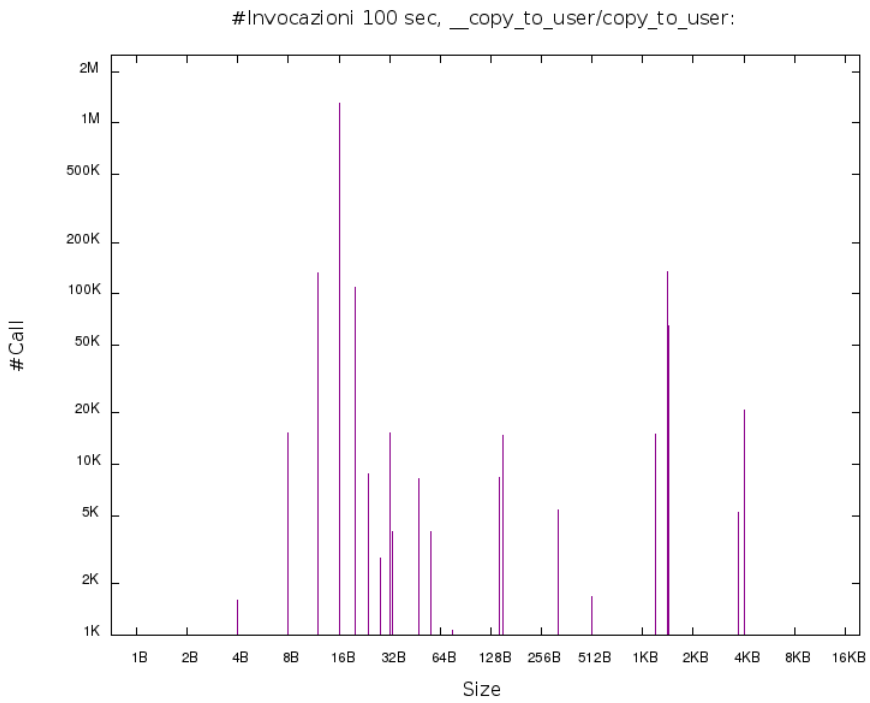

Figura 2.26: Profiling copy_to_user() e __copy_to_user()

2.5 Mibench Performance

Questo paragrafo sarà dedicato all'analisi dei cache miss e dei tempi d'esecuzione di alcuni applicativi del benchmark Mibench. Ogni applicativo sarà eseguito un certo numero di volte su determinate configurazioni, questo ci consentirà di valutare gli effetti di tali configurazioni sulle performance dei programmi. Ricordiamo che il test è eseguito su un sistema che espone 2CPU (2 core che condividono una cache L2).
Le configurazioni prese in esame saranno:

1. **Ideal System**: in questa configurazione si eseguirà solo il programma del Mibench sulla CPU 1, dovrebbe rappresentare l'ottimo in termini di perfomance per l'applicazione in esame.

2. **CPU System**: in questa configurazione si eseguirà il programma del Mibench sulla CPU 1 ed in parallelo sulla CPU 0 sarà lanciato un demone che effettuerà determinate operazioni sulla memoria via CPU.

3. **DMA System**: in questa configurazione si eseguirà il programma del Mibench sulla CPU 1 ed in parallelo sulla CPU 0 sarà lanciato un demone che effettuerà determinate operazioni sulla memoria via DMA.

I test del Mibench che saranno oggetto di studio sono elencati nella tabella sottostante.

Name	**Test Mibench**	**Input**	**L2 Ref.**
susan[s]_small	Susan Smoothing	immagine (~ 7 KB)	1431
susan[e]_small	Susan Edge	immagine (~ 7 KB)	1986
susan[c]_small	Susan Corners	immagine (~ 7 KB)	1652
susan[s]_large	Susan Smoothing	immagine (~ 110 KB)	7776
susan[e]_large	Susan Edge	immagine (~ 110 KB)	39514
susan[c]_large	Susan Corners	immagine (~ 110 KB)	20955
susan[s]_huge	Susan Smoothing	immagine (~ 1 MB)	88063
susan[e]_huge	Susan Edge	immagine (~ 1 MB)	304382
susan[c]_huge	Susan Corners	immagine (~ 1 MB)	108839
qsort_small	Qsort	10^4 parole	157450
qsort_large	Qsort	$5 \cdot 10^4$ punti (x,y,z)	519904

Al fine di rendere questa analisi significativa utilizziamo i dati del sistema di profiling per individuare le coppie funzione-blocco di memoria invocate più spesso, che nel nostro caso sono:

- copy_from_user() e copy_to_user() su blocchi di memoria da 16 Byte
- copy_from_user() su blocchi di memoria da 224 Byte
- copy_from_user() su blocchi di memoria da 1124 Byte
- copy_from_user() su blocchi di memoria da 16064 Byte
- copy_to_user() su blocchi di memoria da 1448 Byte
- copy_to_user() su blocchi di memoria da 4096 Byte

2.5.1 Cache miss

Per ogni test del Mibench sono state effettuate 300 misurazioni, il valore graficato della percentuale di cache miss è la media di queste misurazioni.

Le Figure 2.27 e 2.28 mostrano le percentuali di cache miss relative agli applicativi del Mibench analizzati. In parallelo ove previsto saranno eseguite le operazioni di copy_from_user() e copy_to_user() su blocchi di memoria da 16 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

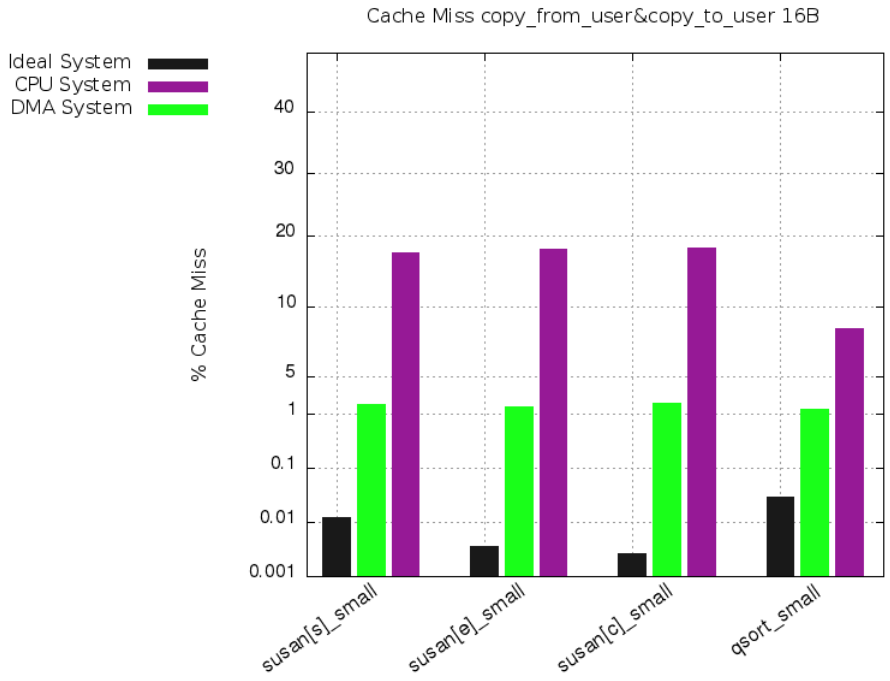

Figura 2.27: Cache Pollution tramite copy_from-to_user() su blocchi da 16 Byte, small test

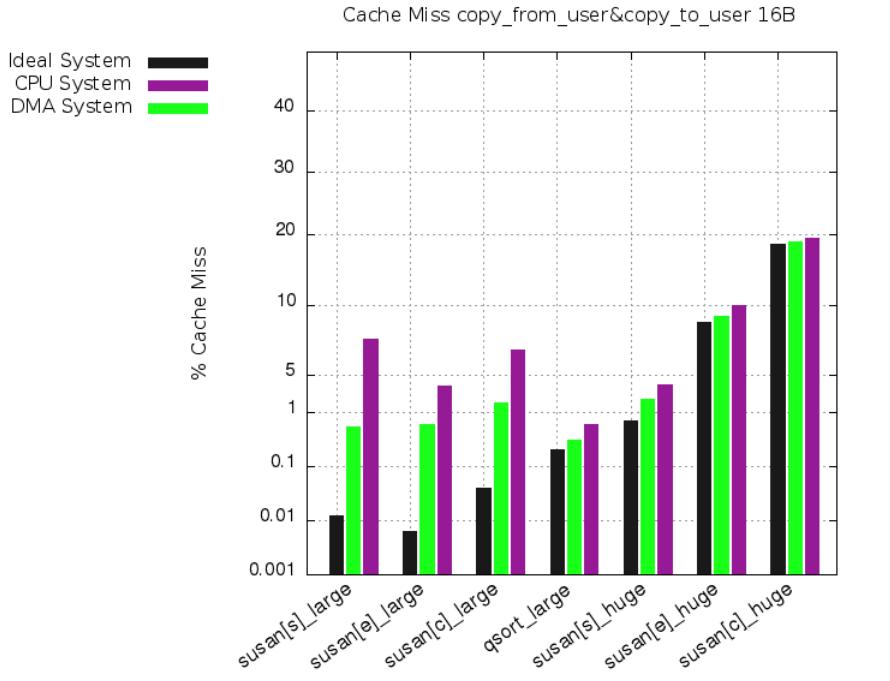

Figura 2.28: Cache Pollution tramite copy_from-to_user() su blocchi da 16 Byte, large test

Le Figure 2.29 e 2.30 mostrano le percentuali di cache miss relative agli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_from_user() su blocchi di memoria da 224 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

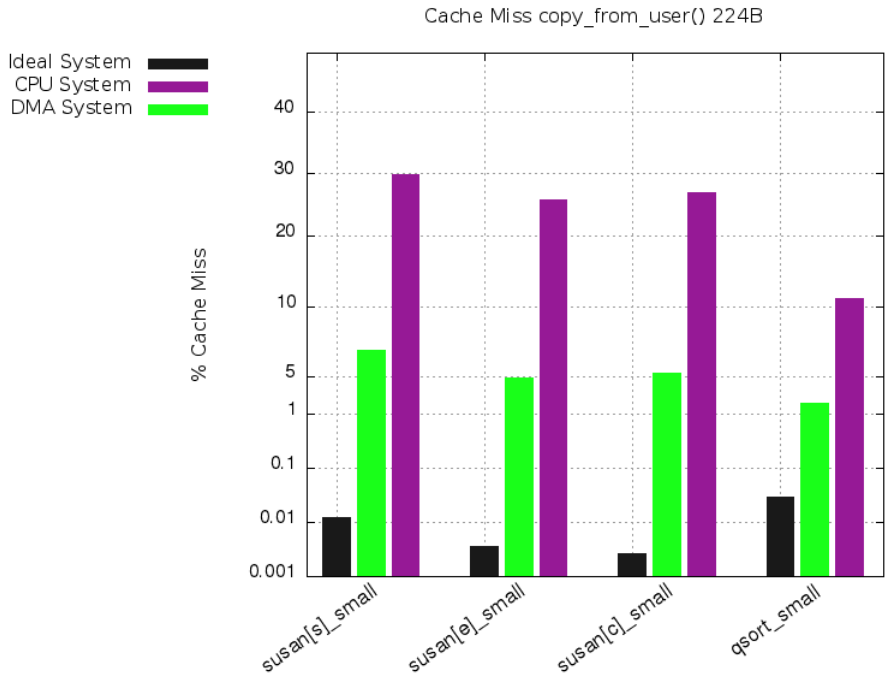

Figura 2.29: Cache Pollution tramite copy_from_user() su blocchi da 224 Byte, small test

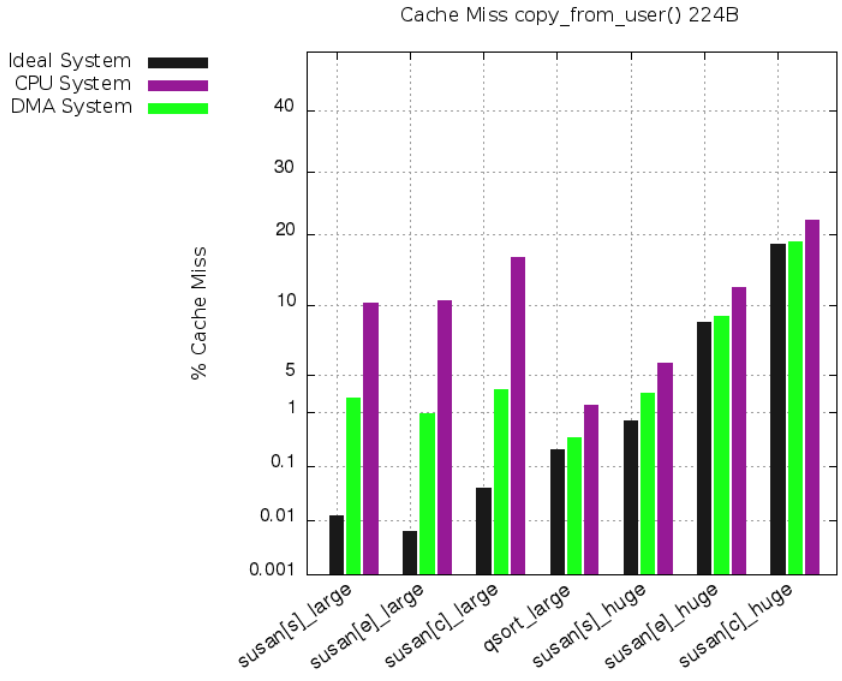

Figura 2.30: Cache Pollution tramite copy_from_user() su blocchi da 224 Byte, large test

Le Figure 2.31 e 2.32 mostrano le percentuali di cache miss relative agli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_from_user() su blocchi di memoria da 1124 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

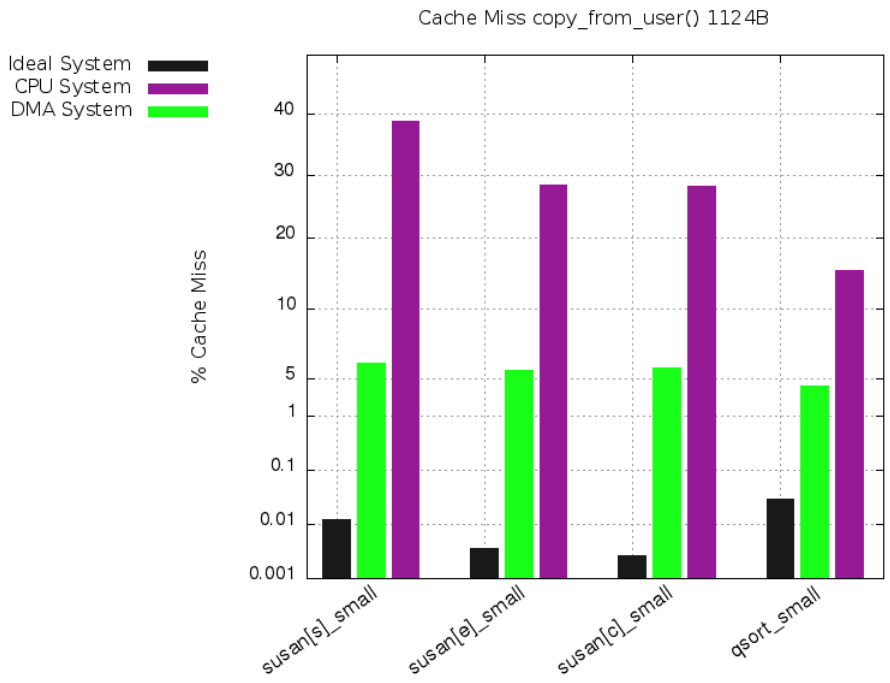

Figura 2.31: Cache Pollution tramite copy_from_user() su blocchi da 1124 Byte, small test

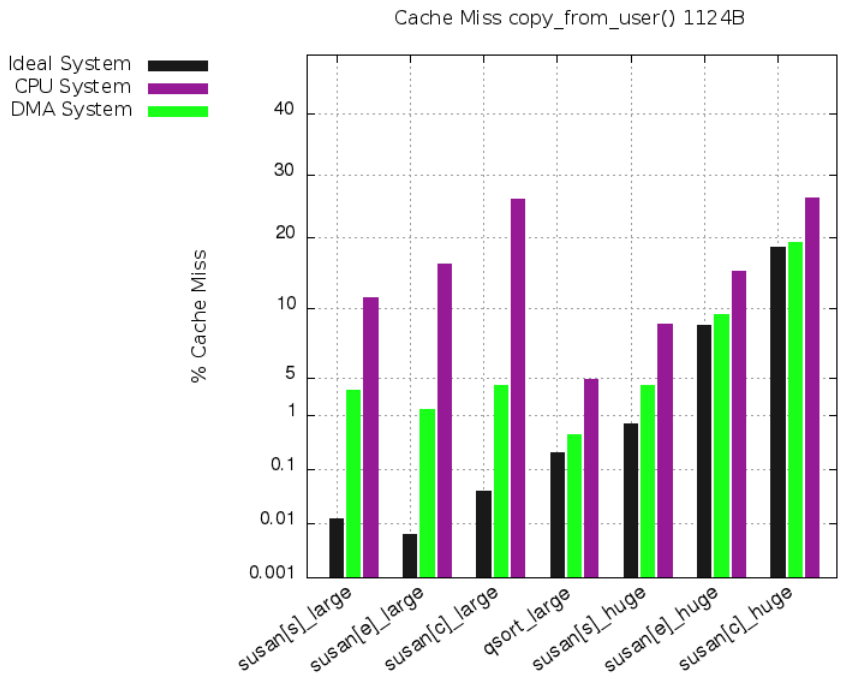

Figura 2.32: Cache Pollution tramite copy_from_user() su blocchi da 1124 Byte, large test

Le Figure 2.33 e 2.34 mostrano le percentuali di cache miss relative agli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_from_user() su blocchi di memoria da 16064 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

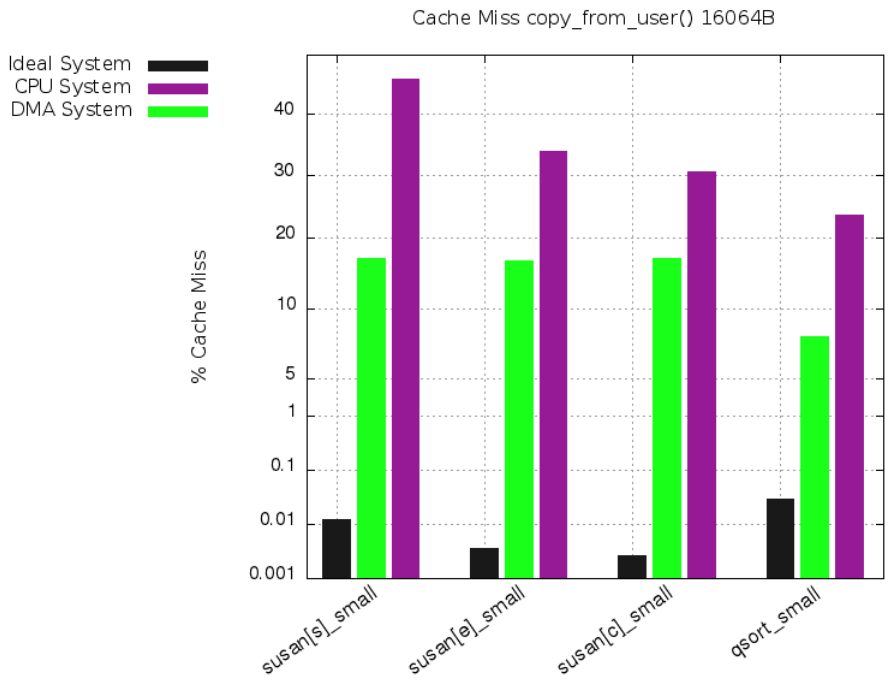

Figura 2.33: Cache Pollution tramite copy_from_user() su blocchi da 16064 Byte, small test

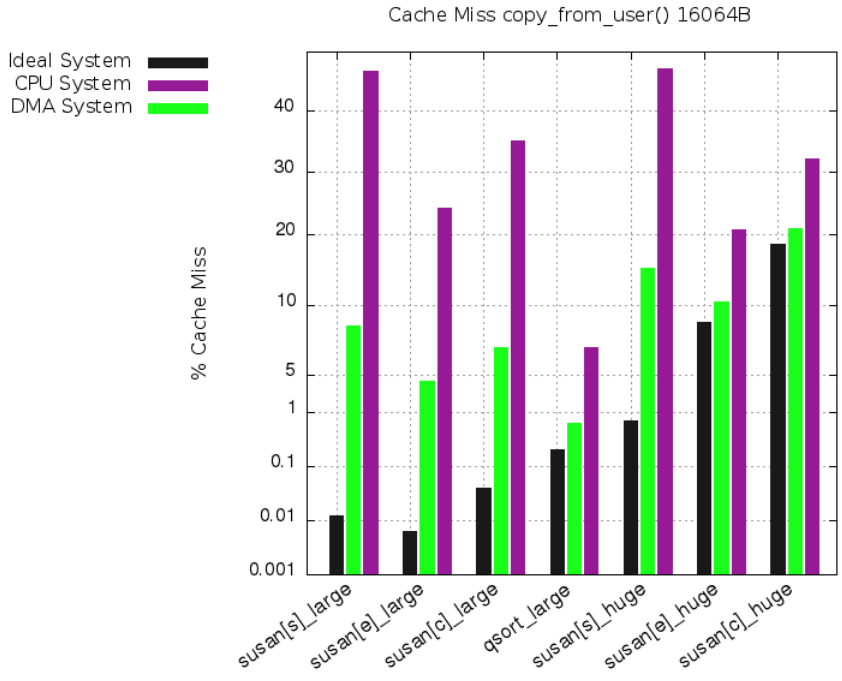

Figura 2.34: Cache Pollution tramite copy_from_user() su blocchi da 16064 Byte, large test

Le Figure 2.35 e 2.36 mostrano le percentuali di cache miss relative agli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_to_user() su blocchi di memoria da 1448 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

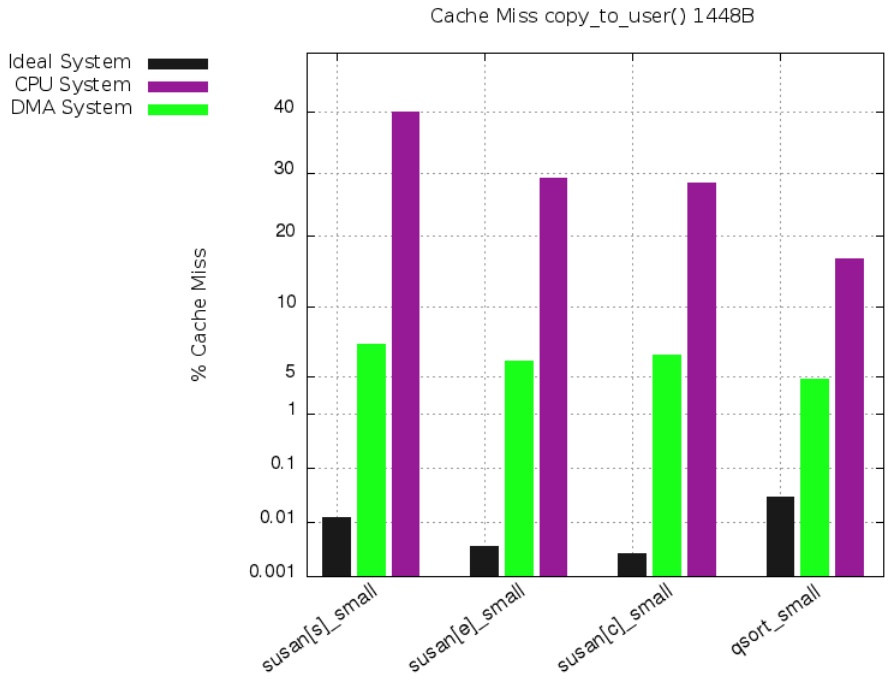

Figura 2.35: Cache Pollution tramite copy_to_user() su blocchi da 1448 Byte, small test

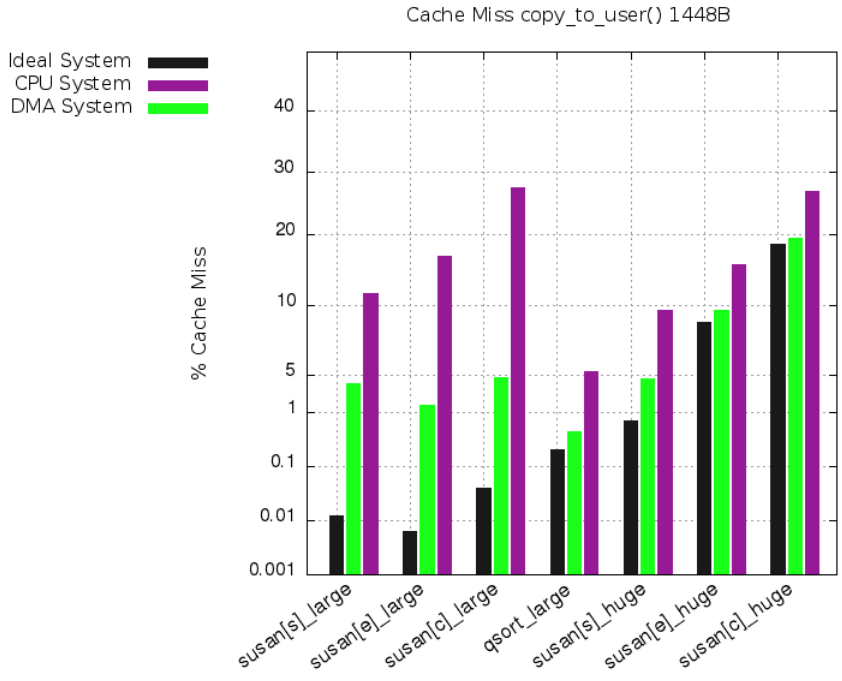

Figura 2.36: Cache Pollution tramite copy_to_user() su blocchi da 16064 Byte, large test

Le Figure 2.37 e 2.38 mostrano le percentuali di cache miss relative agli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_to_user() su blocchi di memoria da 4096 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

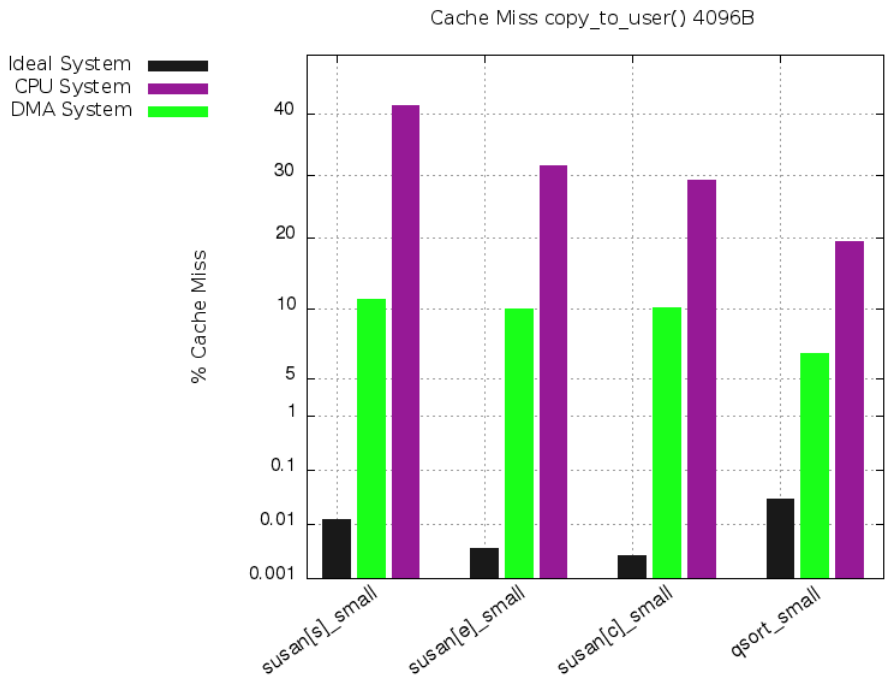

Figura 2.37: Cache Pollution tramite copy_to_user() su blocchi da 4096 Byte, small test

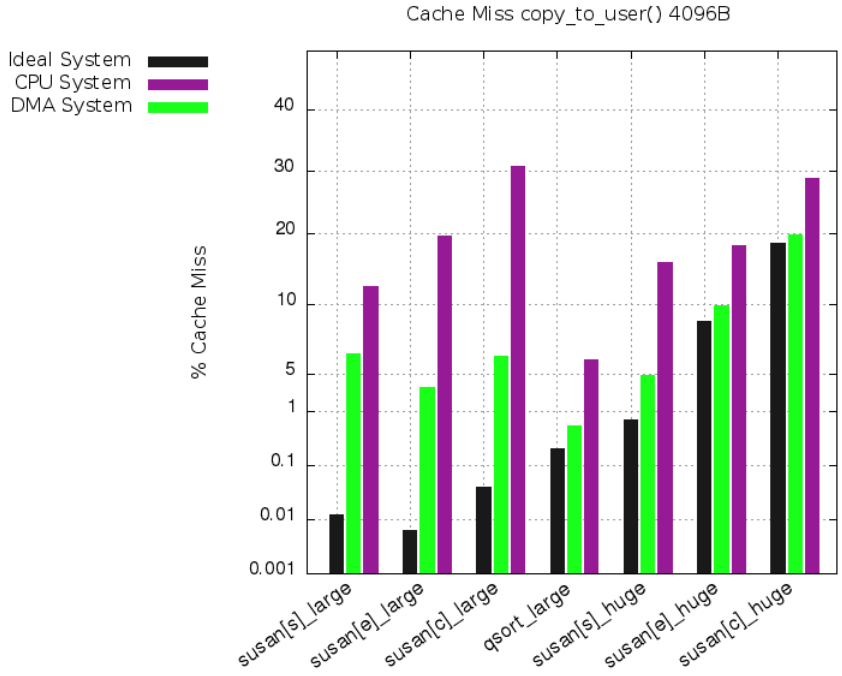

Figura 2.38: Cache Pollution tramite copy_to_user() su blocchi da 4096 Byte, large test

2.5.2 Execution Time

Per ogni test del Mibench sono state effettuate 300 misurazioni, il valore graficato del tempo di esecuzione è la media di queste misurazioni.

Le Figure 2.39 e 2.40 mostrano i tempi di esecuzione degli applicativi del Mibench analizzati. In parallelo ove previsto saranno eseguite le operazioni di copy_from_user() e copy_to_user() su blocchi di memoria da 16 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

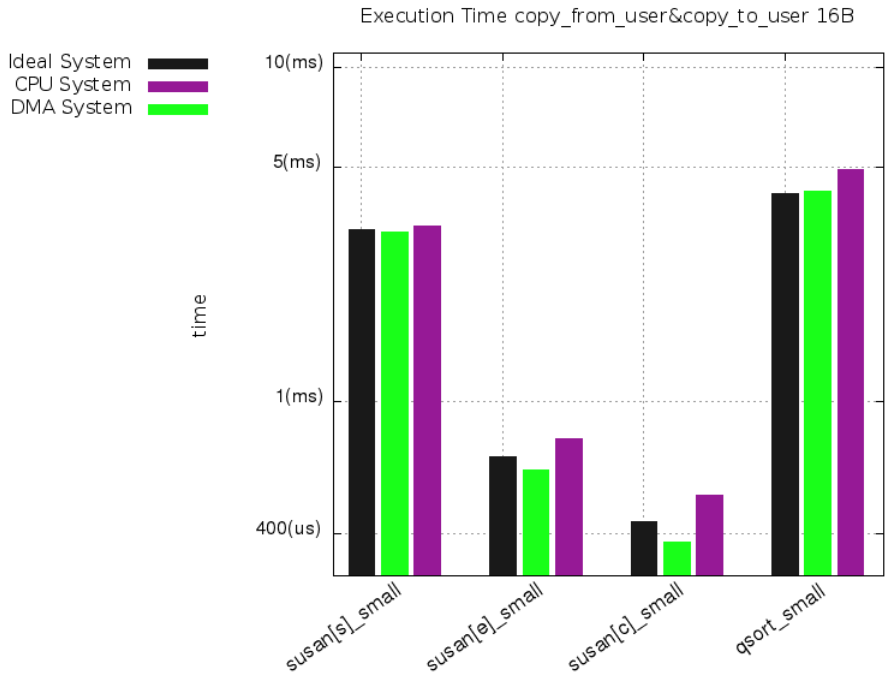

Figura 2.39: Execution Time Mibench (CPU 1), copy_from-to_user() su blocchi da 16 Byte (CPU 0), small test

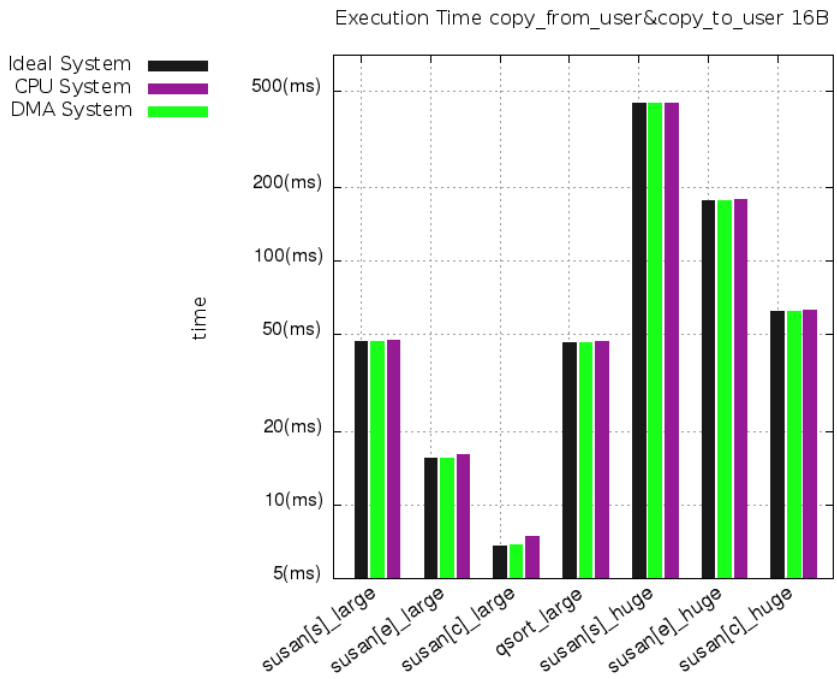

Figura 2.40: Execution Time Mibench (CPU 1), copy_from-to_user() su blocchi da 16 Byte (CPU 0), large test

Le Figure 2.41 e 2.42 mostrano i tempi di esecuzione degli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_from_user() su blocchi di memoria da 224 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

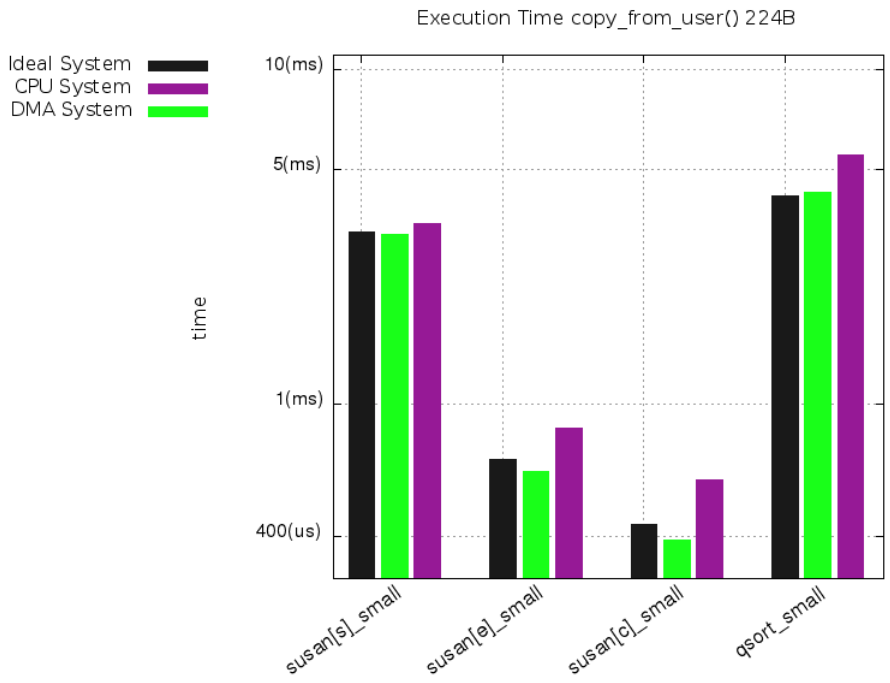

Figura 2.41: Execution Time Mibench (CPU 1), copy_from_user() su blocchi da 224 Byte (CPU 0), small test

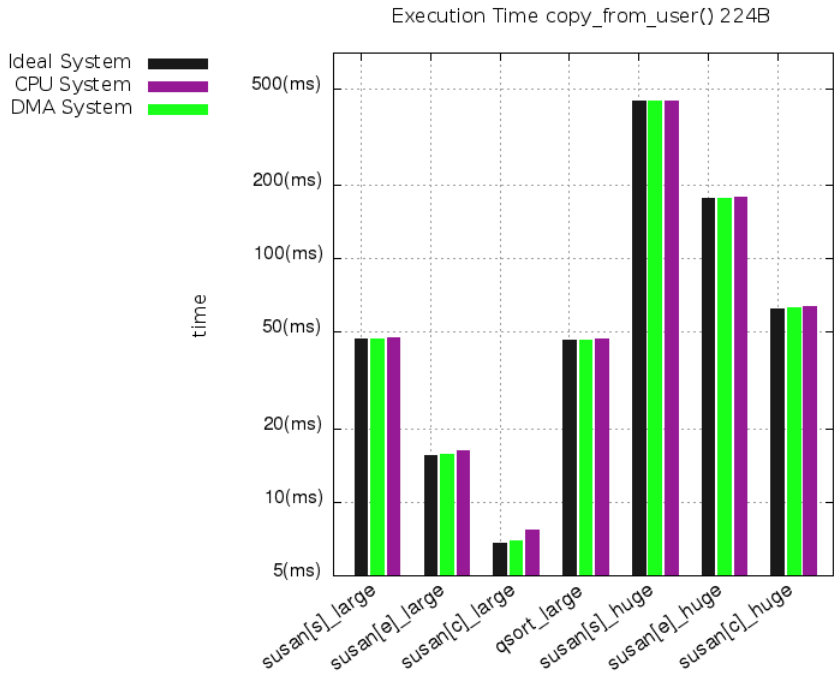

Figura 2.42: Execution Time Mibench (CPU 1), copy_from_user() su blocchi da 224 Byte (CPU 0), large test

Le Figure 2.43 e 2.44 mostrano i tempi di esecuzione degli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_from_user() su

blocchi di memoria da 1124 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

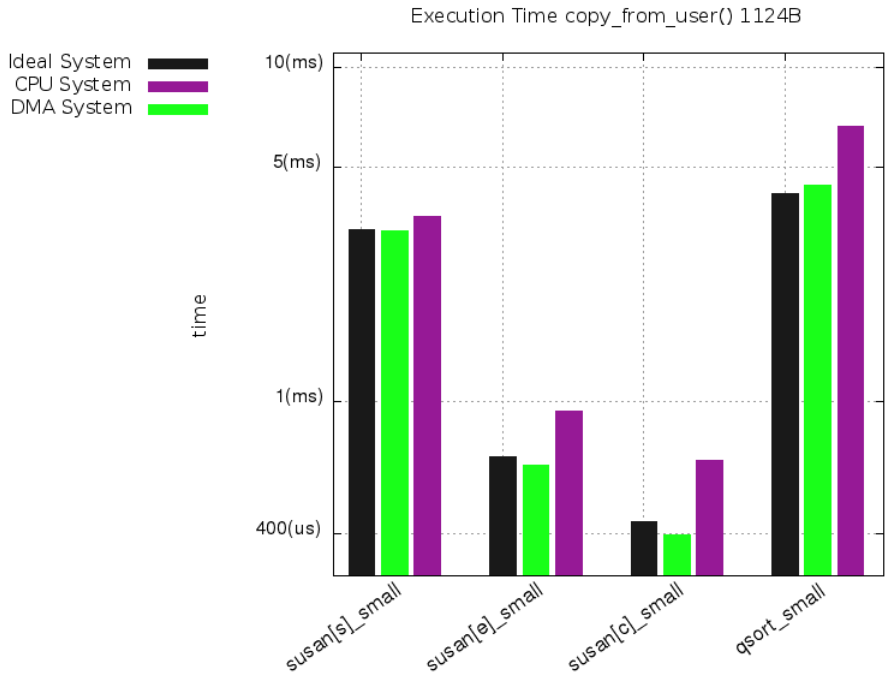

Figura 2.43: Execution Time Mibench (CPU 1), copy_from_user() su blocchi da 1124 Byte (CPU 0), small test

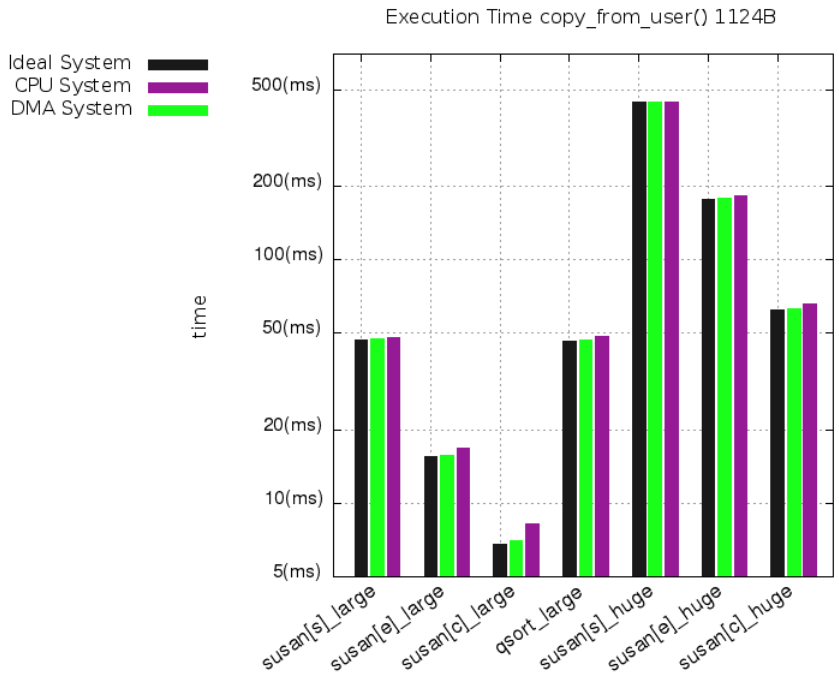

Figura 2.44: Execution Time Mibench (CPU 1), copy_from_user() su blocchi da 1124 Byte (CPU 0), large test

Le Figure 2.45 e 2.46 mostrano i tempi di esecuzione degli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_from_user() su blocchi di memoria da 16064 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

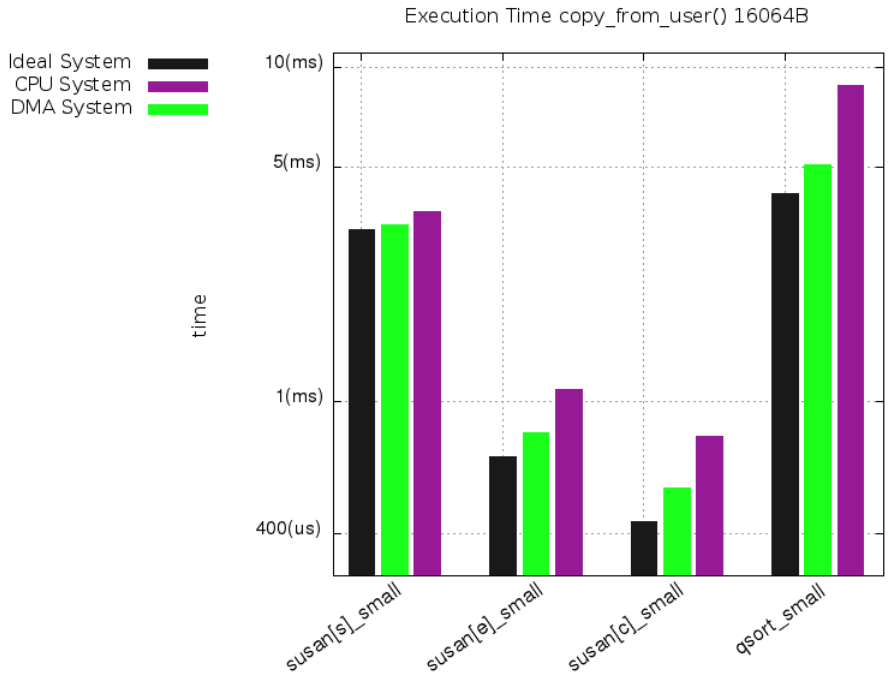

Figura 2.45: Execution Time Mibench (CPU 1), copy_from_user() su blocchi da 16064 Byte (CPU 0), small test

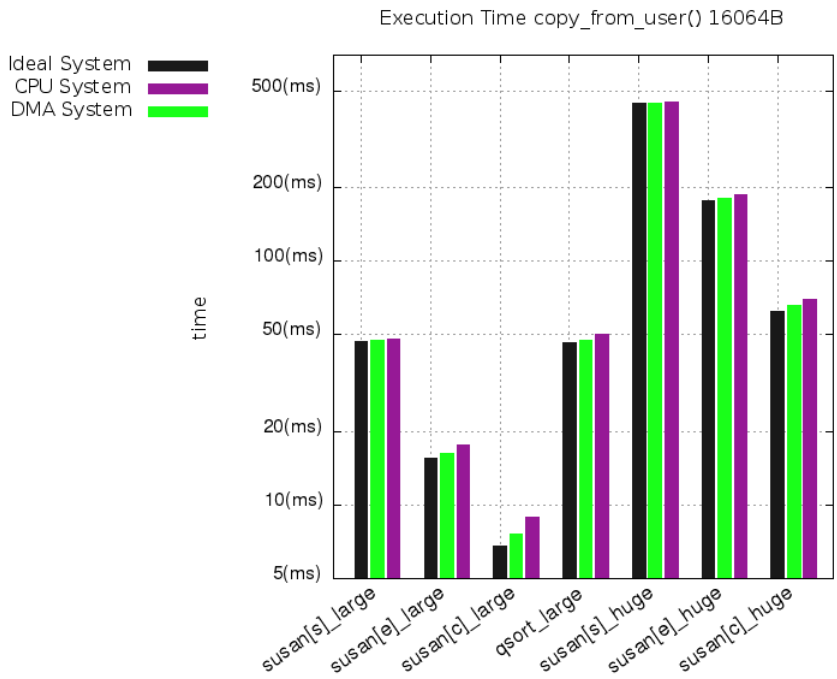

Figura 2.46: Execution Time Mibench (CPU 1), copy_from_user() su blocchi da 16064 Byte (CPU 0), large test

Le Figure 2.47 e 2.48 mostrano i tempi di esecuzione degli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_to_user() su blocchi di memoria da 1448 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

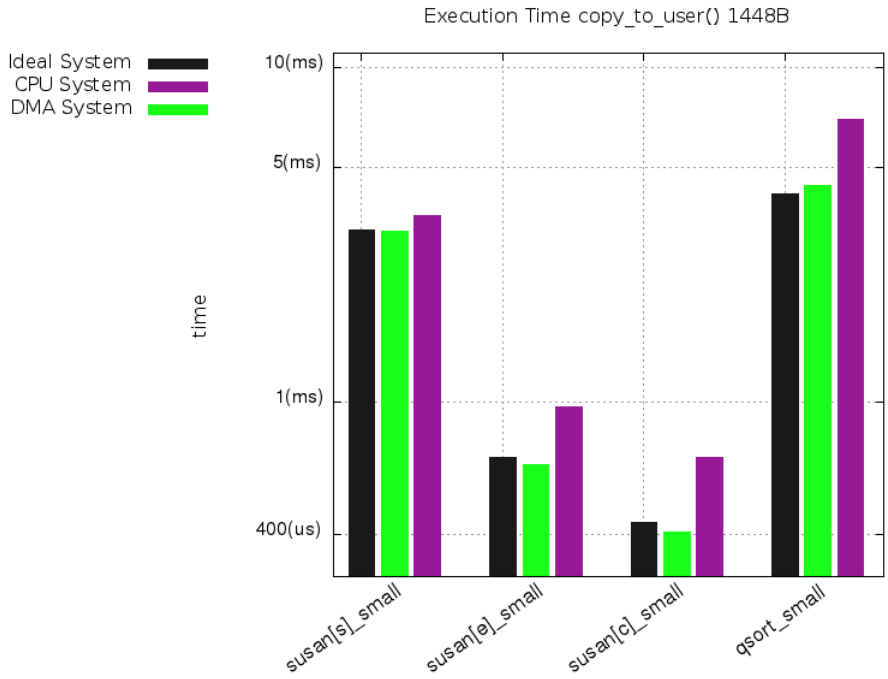

Figura 2.47: Execution Time Mibench (CPU 1), copy_to_user() su blocchi da 1448 Byte (CPU 0), small test

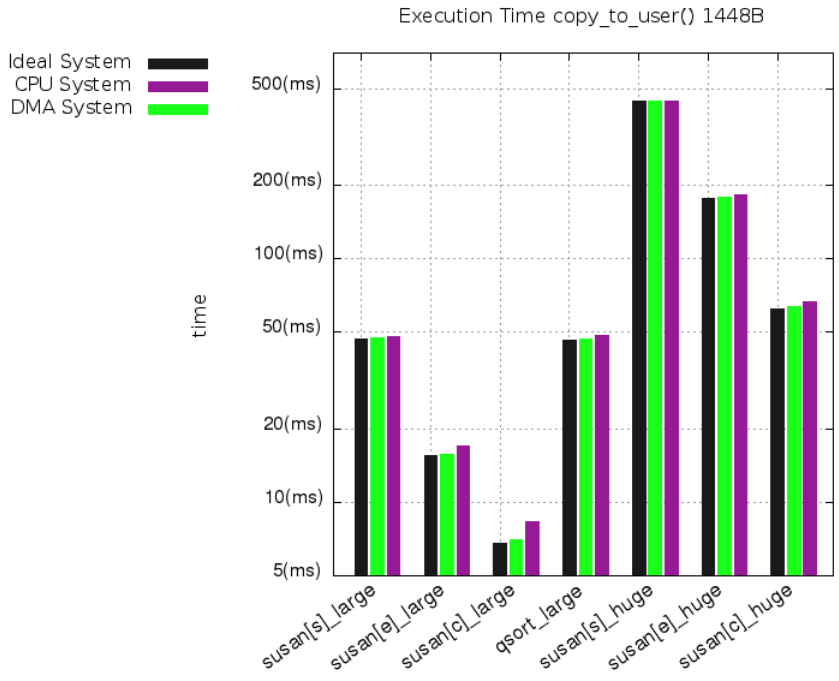

Figura 2.48: Execution Time Mibench (CPU 1), copy_to_user() su blocchi da 1448 Byte (CPU 0), large test

Le Figure 2.49 e 2.50 mostrano i tempi di esecuzione degli applicativi del Mibench analizzati. In parallelo ove previsto sarà eseguita l'operazione di copy_to_user() su

blocchi di memoria da 4096 Byte. Il test è stato ripetuto sulle diverse configurazioni precedentemente elencati.

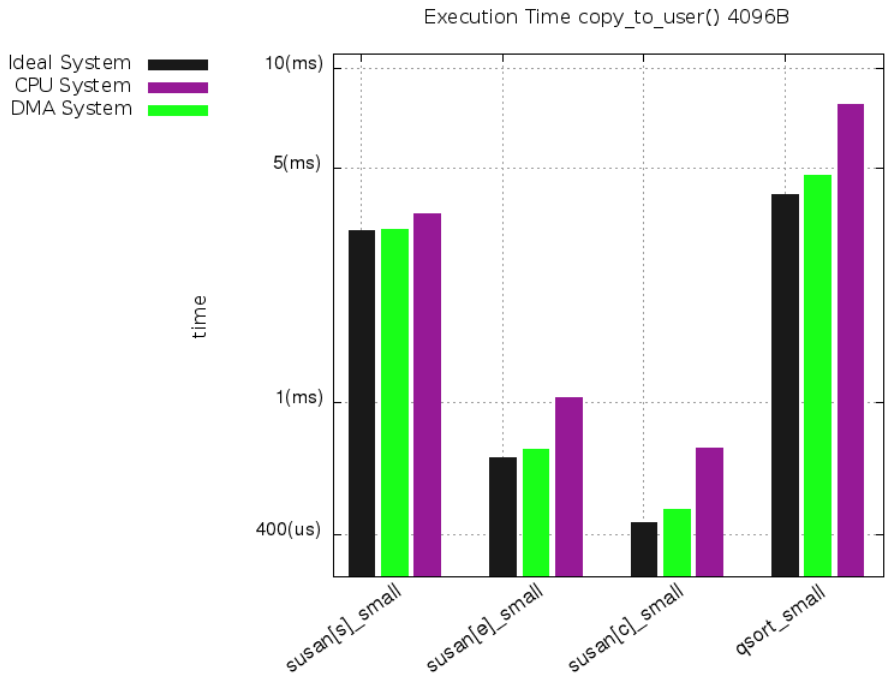

Figura 2.49: Execution Time Mibench (CPU 1), copy_to_user() su blocchi da 4096 Byte (CPU 0), small test

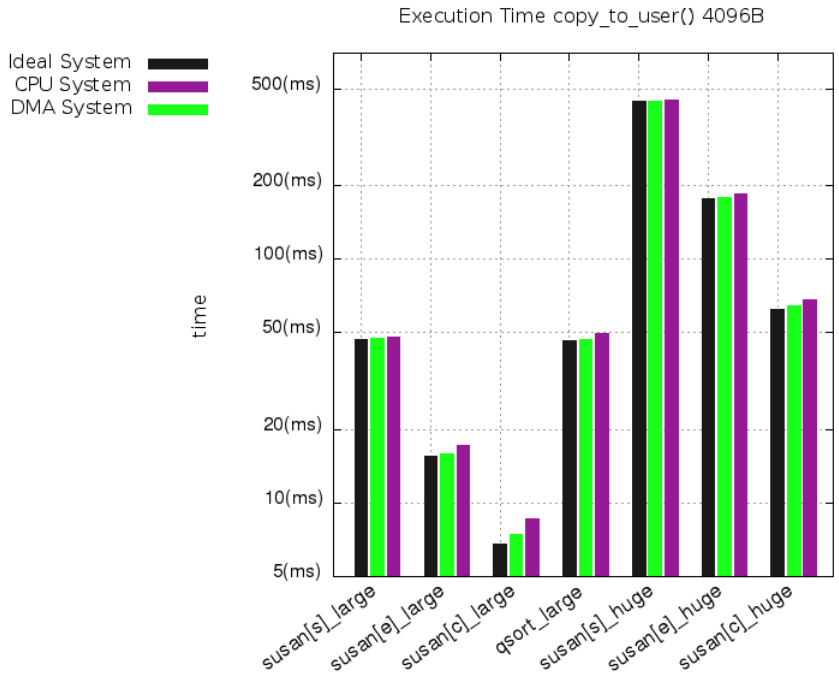

Figura 2.50: Execution Time Mibench (CPU 1), copy_to_user() su blocchi da 4096 Byte (CPU 0), large test

Capitolo 3

Guida all'installazione

In questo capitolo verrà spiegato come configurare la propria macchina al fine di supportare:

- la tecnologia **I/OAT-CFTU**
- il sistema di profiling sulle operazioni di copia in memoria
- il micro-benchmark sulla latenza
- il micro-benchmark sulla larghezza di banda
- il micro-benchmark sulla cache pollution ed execution time

E' bene specificare che non tutte le architetture possono supportare la tecnologia I/OAT-CFTU, infatti il motore DMA dedicato alle copie memoria-memoria è integrato nei seguenti chip:

- Intel® 5000 series chipset integrated device – 1A38
- Intel® 5400 chipset QuickData Technology device - 402F
- Intel® 7300 chipset QuickData Technology device - 360B
- Intel® QuickData Technology device - 65FF

Nel seguito sarà descritta la procedura di setup del sistema relativa all'architettura Apple Xserve.

3.1 Kernel Setup & I/OAT-CFTU Config

Per prima cosa è necessario reperire un kernel vanilla 3.9.2 disponibile al seguente indirizzo: *www.kernel.org/pub/linux/kernel/*.
Applichiamo al suddetto kernel le seguenti patch:

1. **patch-3.9.2.1-xserve** (disponibile nella directory 'patch-kernel')
2. **patch-3.9.2-ioat-cftu** (disponibile nella directory 'patch-kernel')

riportiamo i comandi per queste operazioni:

```
# cd /usr/src/linux-3.9.2
# patch -p1 < ../patch-3.9.2.1-xserve
# patch -p1 < ../patch-3.9.2-ioat-cftu
```

Copiamo il file 'config-3.9.2-ioat-cftu' (disponibile nella directory 'patch-kernel') nei sorgenti del kernel rinominandolo in '.config':

```
# cd /usr/src/linux-3.9.2
# mv config-3.9.2-ioat-cftu ./.config
```

Adesso è necessario modificare il file *.config* per abilitare le funzionalità da noi richieste. In questa trattazione sono state usate 5 configurazioni diverse:

1. **Profiling CPU:** utilizzata per fare il profiling delle copie in memoria fatte dalla CPU
2. **Profiling IOAT-CFTU Excl:** utilizzata per fare il profiling delle copie in memoria fatte dal DMA con Static Policy Exclusive Channels.
3. **Profiling IOAT-CFTU Sh:** utilizzata per fare il profiling delle copie in memoria fatte dal DMA con Static Policy Shared Channels.
4. **Dinamic Policy IOAT-CFTU:** utilizzata per i micro-benchamrk, l'uso delle dynamic policy permette di non definire a priori una politica per il DMA.
5. **Dinamic Policy IOAT-CFTU 2CPU:** utilizzata per il micro-benchamrk sulla cache, la configuraziona è del tutto identica a **Dinamic Policy IOAT-CFTU** l'unica differenza consiste nel forzare il sistema operativo a lavorare solo su 2 CPU.

Di seguito sono riportate le funzionalità da abilitare nel config per le 5 configurazioni sopra elencate.

1. **Profiling CPU** abilitare:

```
1: Kernel hacking->Trace copy_to_user, copy_from_user[*]
```

2. **Profiling IOAT-CFTU Excl** abilitare:

```
1: Device Drivers->DMA Engine support->policy for Intel I/OAT DMA[*]
2: Device Drivers->DMA Engine support->policy for Intel I/OAT DMA
    channel->copy_to/from_user whit I/OAT (exclusive DMA channels)[*]
3: Kernel hacking->Trace copy_to_user, copy_from_user[*]
```

3. **Profiling IOAT-CFTU Sh** abilitare:

```
1: Device Drivers->DMA Engine support->policy for Intel I/OAT DMA[*]
2: Device Drivers->DMA Engine support->policy for Intel I/OAT DMA
    channel->copy_to/from_user whit I/OAT (shared DMA channels)[*]
3: Kernel hacking->Trace copy_to_user, copy_from_user[*]
```

4. **Dinamic Policy IOAT-CFTU** abilitare:

```
1: Device Drivers->DMA Engine support->policy for Intel I/OAT DMA[*]
2: Device Drivers->DMA Engine support->policy for Intel I/OAT DMA
    channel-> Dynamic channel allocations[*]
3: Kernel hacking->Micro-Benchmark memory region at boot time[*]
4: Kernel hacking->Size Micro-Benchmark memory region at boot time
    ->1024 MB[*]
```

5. **Dinamic Policy IOAT-CFTU 2CPU** abilitare:

```
1: Device Drivers->DMA Engine support->policy for Intel I/OAT DMA[*]
2: Device Drivers->DMA Engine support->policy for Intel I/OAT DMA
    channel-> Dynamic channel allocations[*]
3: Kernel hacking->Micro-Benchmark memory region at boot time[*]
4: Kernel hacking->Size Micro-Benchmark memory region at boot time
    ->1024 MB[*]
```

Configurato il nostro kernel, concludiamo la configurazione del sistema:

```
# cd /usr/src/linux-3.9.2
# make -j<n_cpu>
# make modules_install
# mount mount -tvfat /dev/sda1 /mnt/hd
# cat /usr/src/linux-3.9.2/arch/x86/boot/bzImage > /mnt/hd/efi/linux/vmlinuz
    -3.9.2
# (modifichiamo elilo)
# reboot
```

Nella configurazione **Dinamic Policy IOAT-CFTU 2CPU** è necessario aggiungere al record di elilo il seguente *boot time parameter* maxcpus=2.

3.2 Profiling delle operazioni

In questo paragrafo verrà spiegato come effettuare il profiling delle operazioni:

- copy_from_user()
- copy_to_user()
- __copy_from_user()
- __copy_to_user()

L'operazione di profiling può essere effettuata sulle seguenti configurazioni kernel: **Profiling CPU**, **Profiling IOAT-CFTU Excl** e **Profiling IOAT-CFTU Sh**.
Nella directory 'trace' sono disponibili tutti gli strumenti per eseguire il profiling.
In prima istanza lanciamo il driver (shell 1) per il profiling:

```
# cd trace/
# ./start_driver.sh
```

e tramite comando 'dmesg' leggiamo le informazioni scritte dal modulo, in questo caso siamo interessati al numero di CPU utilizzate dal driver. Ecco un esempio di quanto detto precedentemente:

```
# dmesg
..........
mem_trace: Start Module
mem_trace: run on 4 CPU
mem_trace: register
mem_trace: major number 249
```

come è possibile notare in questo caso il driver utilizza 4 CPU.
Dopo aver lanciato il driver ed essersi annotato il numero di CPU è necessario avviare i 4 demoni (uno per ogni operazione) che eseguiranno la raccolta della misurazioni (shell 2):

```
$ cd trace/
$ ./start_daemon.sh <N_CPU> <tempo_misura> <delay>
```

come è possibile notare lo script adibito all'avvio dei demoni ha 3 parametri in input:

1. **N_CPU:** il numero di CPU gestite dal driver.
2. **tempo_misura:** la stima in secondi di quanto debba durare la misura
3. **delay:** il tempo di delay (in μs) tra due richieste di raccolta dati effettuate dal demone verso il driver

dove il numero delle richieste dati N_{req} demone-driver è calcolabile nel seguente modo: $N_{req} = \frac{tempo_{misura} \cdot 10^6}{delay}$
Per chiarire completamente il concetto è bene fare un esempio: *Si ha un sistema che espone 2 CPU al driver per il profiling, lo scopo è quello di effettuare una misurazione che duri almeno 30 minuti con un intervallo di raccolta dati demone-driver di 0.1 sec. Per effettuare suddetto profiling è necessario lanciare:*

```
$ ./start_daemon.sh 2 1800 100000
```

Appena il lavoro dei demoni è concluso e' necessario lanciare (shell 1):

```
# ./stop_driver.sh
```

per rimuovere il modulo del driver.

3.2.1 Risultati Profiling

Terminate le operazioni descritte precedentemente non rimane che esaminare i file dei risultati che sono locati nella cartella 'trace/risultati/', contenete i seguenti file:

- **__mem_trace_from.dat:** per le misurazioni relative alla __copy_from_user()
- **mem_trace_from.dat:** per le misurazioni relative alla copy_from_user()
- **__mem_trace_to.dat:** per le misurazioni relative alla __copy_to_user()

- **mem_trace_to.dat:** per le misurazioni relative alla copy_to_user()

Ogni record dei suddetti file è così organizzato:

a) l'indice della struttura per_cpu che ha fornito il contatore evento

b) valore del contatore evento

c) la size in byte della memoria copiata

d) la cpu che ha iniziato il setup per l'operazione di copia

e) valore del clock (in ns) prima di iniziare la misurazione

f) valore del clock (in ns) a fine misurazione

g) tempo impiegato (in ns) dalla funzione di copia

h) indirizzo di ritorno della funzione instrumentata

i) il tipo di hardware che ha eseguito l'operazione di copia (CPU o DMA)

Ecco un esempio di record:

```
0  1102  8   0  613326537662  613326537755   93    0xffffffff8144c129  CPU
```

3.3 Micro-Benchmark sulla Latenza

In questo paragrafo verrà spiegato come misurare i tempi di latenza delle seguenti operazioni:

- **copy_from_user**
- **copy_from_user (no cache)**
- **copy_to_user**
- **copy_to_user (no cache)**

Il suddetto micro-benchmark può essere eseguito sulla configurazione kernel **Dinamic Policy IOAT-CFTU**.
Nella directory 'micro-benchmark_Msg' sono disponibili tutti gli strumenti per eseguire le misurazioni sulla latenza. Per prima cosa lanciamo il driver per il micro-benchmark (shell 1):

```
# cd micro-benchmark_Msg/
# ./start_driver.sh
```

Lo script 'start_driver.sh' visualizzerà un menù per permettere all'utente di scegliere uno dei tre contesti hardware-software in cui eseguire le misurazioni:

```
Micro-Benchmark on:
0: CPU
1: DMA (exclusive policy)
2: DMA (shared policy)
:
```

Dopo che l'utente avrà scelto il contesto di suo gradimento lo script lancerà il modulo del driver e terminerà.
Dopo aver lanciato il driver è necessario avviare i demoni (shell 2) adibiti alla raccolta delle misure sulle operazioni sottomesse al driver:

```
$ cd micro-benchmark_Msg/
$ ./start_daemon.sh
```

Lo script 'start_daemon.sh' visualizzerà un menù per permettere all'utente di scegliere la size massima dei blocchi di memoria che saranno coinvolti nelle operazioni di copia.

```
Micro-Benchmark size test:
1)  2 MB
2)  4 MB
3)  8 MB
4)  16 MB
5)  32 MB
6)  64 MB
7)  128 MB
8)  256 MB
9)  512 MB
10) 1024 MB
:
```

Dopo che l'utente avrà scelto la size di suo gradimento lo script lancerà i demoni.
Appena il lavoro dei demoni è concluso (shell 2):

```
Start ./daemon for micro-benchmark_from

End ./daemon for micro-benchmark_from

Start ./daemon for micro-benchmark_to

End ./daemon for micro-benchmark_to

Start ./daemon for micro-benchmark_from_nc

End ./daemon for micro-benchmark_from_nc

Start ./daemon for micro-benchmark_to_nc

End ./daemon for micro-benchmark_to_nc
.... End Micro-benchmark
```

e' necessario lanciare (shell 1):

```
# ./stop_driver.sh
```

per rimuovere il modulo del driver.
Come ultima nota è bene specificare che il numero di operazioni dello stesso tipo eseguite su una data size di memoria può essere modificata solo via codice. Le macro da modificare sono:

- **BUFF_MEASURE** in 'micro-benchmark_Msg/driver/micro_benchmark.h'
- **N_UNIT** in 'micro-benchmark_Msg/daemon/daemon.h'

entrambe le macro devono essere settate allo stesso valore.

3.3.1 Risultati Micro-Benchmark sulla latenza

Terminate le operazioni descritte precedentemente non rimane che esaminare i file dei risultati che sono locati nella cartella 'micro-benchmark_Msg/risultati/', contenete i seguenti file:

- **micro-benchmark_from_<x>.dat:** per le misurazioni relative alla copy_from_user()
- **micro-benchmark_from_nc_<x>.dat:** per le misurazioni relative alla copy_from_user() (no cache)
- **micro-benchmark_to_<x>.dat:** per le misurazioni relative alla copy_to_user()
- **micro-benchmark_to_nc_<x>.dat:** per le misurazioni relative alla copy_to_user() (no cache)
- **micro-benchmark_from_<x>_synth.dat:** sintesi delle misurazioni sulla copy_from_user()
- **micro-benchmark_from_nc_<x>_synth.dat:** sintesi delle misurazioni sulla copy_from_user() (no cache)
- **micro-benchmark_to_<x>_synth.dat:** sintesi delle misurazioni sulla copy_from_user()
- **micro-benchmark_to_nc_<x>_synth.dat:** sintesi delle misurazioni sulla copy_to_user()(no cache)

il campo <x> presente nei nomi dei file, rappresenta il valore scelto dall'utente nel menù generato dallo script 'start_daemon.sh'. Facciamo un esempio: il valore 1 in 'micro-benchmark_from_1.dat' ci indica che l'utente nel menù ha scelto 2 MB come size massima dei blocchi di memoria.
Infine illustriamo la strutturazione dei file presenti nella directory 'risultati':

- I record dei file contenenti le misurazioni sono così organizzati

  a) la cpu che ha effettuato la misurazione temporale tramite tsc

  b) la size in byte della memoria copiata

  c) valore del clock (in ns) prima di iniziare la misurazione

d) valore del clock (in ns) a fine misurazione

e) tempo impiegato (in ns) dall'operazione di copia

Ecco un esempio di record:

```
3   64   2740081346085    2740081346193   108
```

- I record dei file contenenti le sintesi sono così organizzati

  a) la size in byte della memoria copiata

  b) il tempo (in ns) impiegato dall'operazione di copia più veloce

  c) il tempo (in ns) impiegato dall'operazione di copia più lenta

  d) la media campionaria del tempo (in ns) impiegato dalle operazioni di copia

  e) la varianza campionaria dei tempi (in ns) impiegati dalle operazioni di copia

  f) la mediana dei tempi (in ns) impiegati dalle operazioni di copia

  Ecco un esempio di record:

```
1   96   115    98.3300   4.0546   99.0000
```

3.4 Micro-Benchmark sulla Bandwidth

In questo paragrafo verrà spiegato come misurare la bandwidth della CPU e del DMA mentre effettuano le seguenti operazioni di copia:

- **copy_from_user**
- **copy_to_user**
- **copy_from_user + copy_to_user**

Il suddetto micro-benchmark può essere eseguito sulla configurazione kernel **Dinamic Policy IOAT-CFTU**.
Nella directory 'micro-benchmark_Band' sono disponibili tutti gli strumenti per eseguire le misurazioni sulla velocità di trasferimento. Per prima cosa lanciamo il driver per il micro-benchmark (shell 1):

```
# cd micro-benchmark_Band/
# ./start_driver.sh
```

Lo script 'start_driver.sh' visualizzerà un menù per permettere all'utente di scegliere uno dei tre contesti hardware-software in cui eseguire le misurazioni:

```
Micro-Benchmark on:
1: CPU
2: DMA (exclusive policy)
3: DMA (shared policy)
4: DMA (priority policy)
:
```

Dopo che l'utente avrà scelto il contesto di suo gradimento lo script lancerà il modulo del driver e terminerà.
Dopo aver lanciato il driver è necessario avviare il demone (shell 2) adibito alla raccolta delle misure sulle operazioni sottomesse al driver:

```
$ cd micro-benchmark_Msg/
$ ./start_daemon.sh
```

Lo script 'start_daemon.sh' visualizzerà un menù per permettere all'utente di scegliere la size massima dei blocchi di memoria che saranno coinvolti nelle operazioni di copia:

```
Micro-Benchmark size test:
1)  2 MB
2)  4 MB
3)  8 MB
4)  16 MB
5)  32 MB
6)  64 MB
7)  128 MB
8)  256 MB
:
```

ed il numero di operazioni dello stesso tipo da eseguire su un blocco di memoria della stessa size:

```
insert #operation:
```

Terminata l'interazione con l'utente, lo script lancerà il demone.
Appena il lavoro del demone è concluso (shell 2):

```
Start ./daemon

End ./daemon
.... End Micro-benchmark
```

e' necessario lanciare (shell 1):

```
# ./stop_driver.sh
```

per rimuovere il modulo del driver.
Come ultima nota è bene specificare che i quanti temporali per il contesto hardware-software **DMA (priority policy)** possono essere modificati solo via codice. Le macro da modificare sono:

1. **_NANOSEC_HIGH** in 'micro-benchmark_Band/driver/micro_benchmark.h'
2. **_NANOSEC_LOW** in 'micro-benchmark_Band/driver/micro_benchmark.h'

dove _NANOSEC_HIGH indica il quanto di tempo in ns in cui il canale a priorità alta ha l'uso esclusivo del DMA e _NANOSEC_LOW è il quanto di tempo in ns in cui tutti i canali competono per l'uso del DMA.

3.4.1 Risultati Micro-Benchmark sulla Bandwidth

Terminate le operazioni descritte precedentemente non rimane che esaminare i file dei risultati che sono locati nella cartella 'micro-benchmark_Band/risultati/', contenete i seguenti file:

- **measures_from.dat:** per le misurazioni relative alla copy_from_user()
- **measures_from_to.dat:** per le misurazioni relative alla copy_from_user() + copy_to_user()
- **measures_to.dat:** per le misurazioni relative alla copy_to_user()

Ogni record dei suddetti file è così organizzato:

a) id del canale/core che ha eseguito la copia

b) la size in byte della memoria copiata in una operazione

c) il numero di operazioni eseguite

d) valore del clock (in ns) prima di iniziare la misurazione

e) valore del clock (in ns) a fine misurazione

f) tempo impiegato (in ns) per espletare le operazioni di copia

g) la velocità di trasferimento del canale in MB/sec

i) la velocità di trasferimento dell'intero sistema in MB/sec (nota questo campo è inserito solo nella riga corrispondente all'ultimo canale).

Concludiamo riportando alcuni record delle misurazioni per meglio chiarire la strutturazione dei file:

```
0 1024  300  1384270020290328601  1384270020290437344  108743  2694.138933
1 1024  300  1384270020290343080  1384270020290437040  93960   3118.015645
2 1024  300  1384270020290320607  1384270020290426168  105561  2775.350271
3 1024  300  1384270020290336009  1384270020290431531  95522   3067.029061
    total_band= 11654.533911
```

3.5 Micro-Benchmark sulla Cache Pollution

In questo paragrafo verrà spiegato come misurare le performance relative ai cache miss ed execution time di alcune applicazioni del benchmark Mibench (CPU 1):

- **Susan Smoothing**
- **Susan Edge**
- **Susan Corners**
- **Qsort**

mentre sono in esecuzione trasferimenti dati tra memoria utente e kernel (CPU 0).
Il suddetto micro-benchmark è essere eseguito sulla configurazione kernel **Dinamic Policy IOAT-CFTU 2CPU**.
Per prima cosa lanciamo il driver per il micro-benchmark (shell 1):

```
# cd micro-benchmark_Cache_2CPU/
# ./start_driver.sh
```

Lo script 'start_driver.sh' visualizzerà un menù per permettere all'utente di scegliere uno dei due contesti hardware-software in cui verranno eseguiti i trasferimenti dati tra memoria utente e kernel:

```
Micro-Benchmark on:
1: CPU
2: DMA (shared policy)
:
```

Dopo che l'utente avrà scelto il contesto di suo gradimento lo script lancerà il modulo del driver e terminerà.
Dopo aver lanciato il driver è necessario avviare il demone (shell 2) per le operazioni sulla memoria:

```
$ cd micro-benchmark_Cache_2CPU/
$ ./start_daemon.sh
```

Lo script 'start_daemon.sh' visualizzerà un menù per permettere all'utente di scegliere la size dei blocchi di memoria che saranno coinvolti nelle operazioni di copia:

```
 - Micro-Benchmark-Cache 2CPU-
Size buffer memory in byte:
:
```

ed il tipo di operazioni da eseguire:

```
insert  type_operation:
1) copy_from_user()
2) copy_to_user()
3) copy_from_user() + copy_to_user()
:
```

Terminata l'interazione con l'utente, lo script lancerà il demone:

```
Start ./daemon

Press a key to end....
```

Adesso con il demone in esecuzione possiamo lanciare i test del Mibench (shell 3):

```
$ cd micro-benchmark_Cache_2CPU/
$ ./start_mibench.sh
```

Lo script 'start_mibench.sh' visualizzerà un menù per permettere all'utente di scegliere quante volte eseguire lo stesso test:

```
 - MiBench Test -
insert #test
:
```

ed il nome del file in cui salvare le misurazioni:

```
name file:
:
```

Un volta che i test del Mibench saranno conclusi (shell 3):

```
.... End MiBench
```

digitiamo un tasto per concludere il demone (shell 2):

```
End ./daemon
.... End Micro-benchmark
```

ed infine lanciamo (shell 1):

```
# ./stop_driver.sh
```

per rimuovere il modulo del driver.
Come ultima nota è bene specificare che il delay (in usec) tra due test dello stesso tipo è modificabile solo via codice. La macro da modificare è:

- **SLEEP_TIME** in 'micro-benchmark_Cache_2CPU/mibench_automotive/mibench.h'

3.5.1 Risultati Micro-Benchmark sulla Cache Pollution

Terminate le operazioni descritte precedentemente non rimane che esaminare i file dei risultati che sono locati nella cartella 'micro-benchmark_Cache_2CPU/risultati/', contenete i seguenti file:

- **measure_<x>.dat:** le misurazioni relative ai test del Mibench
- **synthesis_<x>.dat:** sintesi delle misurazioni

il campo <x> presente nei nomi dei file, rappresenta il 'name file' scelto dall'utente nel menù generato dallo script 'start_mibench.sh'.
Infine illustriamo la strutturazione dei file presenti nella directory 'risultati':

- I record dei file contenenti le misurazioni sono così organizzati
  - **a)** il tipo di test eseguito
  - **b)** valore del clock (in ns) prima di iniziare la misurazione
  - **c)** valore del clock (in ns) a fine misurazione
  - **d)** tempo impiegato (in ns) per eseguire il test
  - **e)** il numero di accessi alla cache (cache ref) necessari per eseguire il test
  - **f)** il numero di cache miss

g) la percentuale di cache miss

Ecco un esempio di record:

```
qsort_small  55698518635  55702757592  4238957  157283  982  0.624352
```

- I record dei file contenenti le sintesi sono così organizzati

  a) il tipo di test eseguito

  b) il numero di ripetizioni del test

  c) la media campionaria del tempo (in us) d'esecuzione

  d) il maggior tempo (in us) d'esecuzione

  e) la media campionaria dei cache ref

  f) la media campionaria della percentuale dei cache miss

  Ecco un esempio di record:

```
qsort_small 300  4408.6608   4599  157189.8800  2.245550
```