Implementazione di Memoria Distribuita su cluster CompactPCI

IMPLEMENTAZIONE DI MEMORIA DISTRIBUITA SU CLUSTER COMPACTPCI

Maja Massarini*, Marino Miculan**, Francesco Sepic**
*CRSCasa S.r.l.
m.massarini@crscasa.it
**Dipartimento di Matematica e Informatica
Università degli Studi di Udine
miculan@dimi.uniud.it, sepic@dimi.uniud.it

In questo articolo si descrive l'implementazione di una memoria distribuita condivisa (DSM) per un cluster di calcolatori connessi da un backplane CompactPCI, anziché da una normale rete a pacchetti. Questa soluzione hardware e software innovativa consente di implementare librerie di parallelizzazione (come MPI) senza utilizzare lo stack TCP/IP, con evidenti guadagni prestazionali. Inoltre, la memoria condivisa distribuita è utilizzabile direttamente a livello utente, rendendo il multicomputer in esame simile alle architetture NUMA e consentendo così modelli di programmazione semplificati rispetto alla programmazione basata su MPI.

1. Introduzione

La potenza computazionale raggiungibile dalle architetture mono- e multiprocessore a memoria fisica condivisa (come SMP) è intrinsecamente limitata da problemi di dissipazione di calore e dalla complessità costruttiva [11]. Un modo comune per aggirare l'ostacolo è impiegare architetture parallele *senza memoria condivisa*, ed in particolare i *multicomputer* [12]. Ciascun nodo di un multicomputer è un calcolatore a se stante, composto da CPU (singola o SMP), memoria privata, un'interfaccia di rete ed eventualmente un disco rigido; per questi motivi, questi sistemi vengono anche chiamati COW (cluster of workstations).

Naturalmente, le performance di un sistema multicomputer dipendono in maniera cruciale dall'infrastruttura di comunicazione; questa può variare da semplici ed economici NIC Ethernet (come nei sistemi Beowulf [13]), fino ai più costosi dispositivi di comunicazioni *ad hoc*, come ad esempio Myrinet [2].

In questo lavoro, si considera una soluzione innovativa al problema, recentemente proposta dalla Eurotech SpA. La particolarità di questa soluzione è l'uso del bus CompactPCI [8] come infrastruttura di comunicazione, al posto di reti a commutazione o reti dedicate. Attraverso il bus CompactPCI, ogni CPU ha (limitato) accesso alla memoria principale degli altri nodi, rendendo così il sistema parzialmente simile ad una NUMA. Questa soluzione permette di abbattere i tempi di latenza, rimanendo economicamente competitiva.

Come per altri multicomputer, il modo naturale per programmare questa soluzione hardware è attraverso le librerie MPI, che costituiscono uno standard *de facto* del settore [9]. Molte implementazioni di MPI prevedono la possibilità di usare canali TCP/IP per la comunicazione tra i nodi (ad esempio su NIC Ethernet; Figura 1(a)); quindi, il *porting* di MPI all'architettura CompactPCI può

essere realizzato implementando delle “schede di rete virtuali” su tale bus, sulle quali appoggiare il resto dello stack di rete (Figura 1(b)). Driver di rete di questo tipo sono disponibili in distribuzioni Linux specializzate [7].

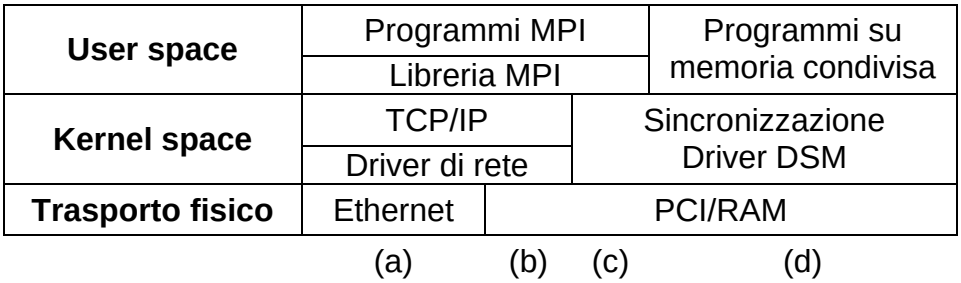

Figura 1: Possibili stack di comunicazione.

Questa soluzione non è del tutto soddisfacente, in quanto la complessità dello stack TCP/IP introduce un overhead eccessivo rispetto alle esigenze di indirizzamento di una rete CompactPCI. Si vuole quindi rimuovere lo stack TCP/IP ed implementare le MPI direttamente, mediante l’accesso alla memoria remota realizzata attraverso il bus CompactPCI, come in Figura 1(c).
Il lavoro descritto in questo articolo è il primo passo in questa direzione. Precisamente, si descrive l’implementazione di una *memoria distribuita condivisa* (DSM) sul bus CompactPCI, per il sistema operativo Linux. Con questa funzionalità, i processi di un dato nodo possono accedere ai segmenti di memoria dei nodi remoti usando le API standard dei file: lettura, scrittura e mappatura in memoria. Inoltre, il driver implementa anche le funzionalità di sincronizzazione distribuita, necessarie per il controllo di accesso ad una memoria condivisa.
La memoria distribuita condivisa così sviluppata può essere impiegata in due modi. In primo luogo, è possibile usare questa memoria condivisa per implementare direttamente le MPI come in Figura 1(c), senza passare per lo stack di rete; ad esempio, sviluppando un adeguato modulo per LAM/MPI ([3], [10]). Alternativamente, è possibile usare il driver direttamente, senza passare affatto attraverso le librerie MPI, come schematizzato in Figura 1(d). Secondo questo approccio, si offre la possibilità di un modello di programmazione a memoria condivisa, anziché a scambio di messaggi. In questo lavoro, viene mostrata quest’ultima applicazione, lasciando la prima come sviluppo futuro.

Sinossi Nella sezione 2 viene descritta l’architettura hardware di riferimento. L’implementazione del driver DSM e delle primitive di sincronizzazione sono descritte in sezione 3. Una valutazione delle performance è presentata in sezione 4. Alcune conclusioni e direzioni per lavoro futuro sono riportate nella sezione 5. L’Appendice A descrive brevemente le API per l’accesso alle funzionalità del driver.

2. Architettura hardware

Il bus di comunicazione è il punto debole di ogni architettura parallela poiché esso rappresenta il principale limite alla sua scalabilità. Nelle architetture multicomputer, con rete di interconnessione basata su bus, ogni processore è dotato di un bus locale sul quale grava la maggior parte del traffico prodotto dalla CPU. In questo modo il carico di lavoro destinato al bus di sistema, che è quello sul quale comunicano le diverse CPU dell'elaboratore, risulta inferiore.
Questo schema di bus decentralizzati permette di ottenere una maggiore scalabilità rispetto ad un sistema multiprocessore di tipo UMA connesso tramite bus oppure una scalabilità paragonabile ad un sistema NUMA connesso da bus ma ad un costo decisamente inferiore.

2.1. CompactPCI

Il CompactPCI (cPCI) [8] è un'estensione del PCI, rispondente ai requisiti di solidità e robustezza dei sistemi embedded ed industriali. Aderisce alle specifiche elettriche dei comuni bus PCI, garantendo piena compatibilità con il PCI 2.0, tuttavia differenziandosi da questi in primo luogo per la diversa tipologia di connettori, che permettono di inserire e rimuovere le schede più accuratamente, e garantiscono prestazioni superiori per quanto riguarda la trasmissione del segnale e la schermatura dei rumori. Viene supportato pienamente anche l'hot swapping, caratteristica fondamentale soprattutto nei sistemi fault tolerant.

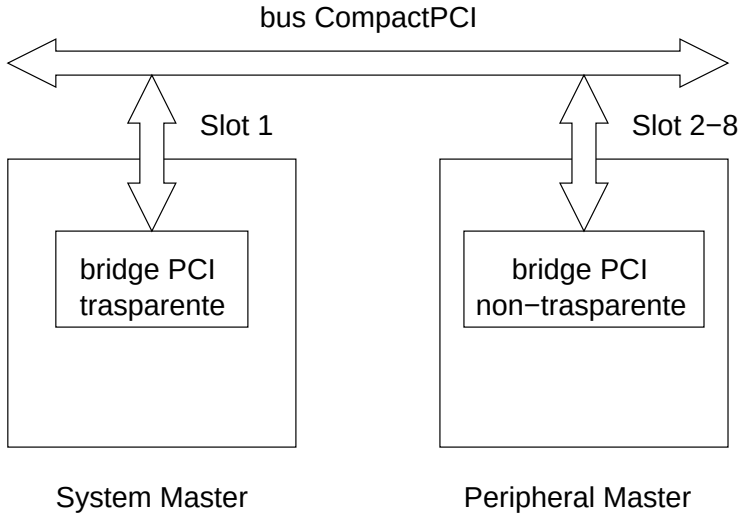

Figura 2: Struttura di un sistema CompactPCI.

Un sistema CompactPCI è composto da uno o più segmenti di bus (Figura 2). Ogni segmento può ospitare fino a otto schede CompactPCI. Un slot ospita il System Master, mentre negli altri sette slot ci possono essere dei Peripheral Master. Ciò che differenzia elettricamente e fisicamente queste schede è il bridge PCI. Il System Master infatti è equipaggiato con un PCI-to-PCI bridge, mentre i Peripheral Master montano un bridge non-trasparente (chiamato anche Embedded). Questo bridge viene inserito direttamente sulla scheda CompactPCI.

2.2. Il bridge PCI-to-PCI Intel 21554

Il cuore di questo sistema multicomputer è il bridge PCI non-trasparente dei Peripheral Master. Questo tipo di bridge realizza due importanti funzioni: suddivide i bus PCI connessi in bus PCI primario e bus PCI secondario (bus locale interno alla scheda Peripheral Master) e gestisce lo scambio di dati tra i due bus PCI in termini di memoria condivisa.
Nel sistema in esame, il bridge PCI è un Intel 21554, il cui utilizzo trova maggior impiego nell'ambito delle applicazioni embedded [5]. Questo dispositivo è un bridge PCI-to-PCI non-trasparente a 64 bit, che si comporta da gateway in un sottosistema intelligente (ossia, un qualsiasi sistema dotato di unità di calcolo). Permette inoltre al processore locale di configurare e controllare il sottosistema locale in maniera indipendente. Siccome il 21554 è indipendente dall'architettura, funziona con qualsiasi processore che supporta il bus PCI.
Questo dispositivo permette di inoltrare le transazioni tra il bus PCI primario e quello secondario proprio come un bridge trasparente. La differenza è che può convertire gli indirizzi della transazione da indirizzi di host a indirizzi locali e vice versa. Questo meccanismo permette al 21554 di nascondere le risorse tra i due sistemi e risolvere i conflitti che si potrebbero generare.
Esso implementa dei base address register (BAR) separati su entrambe le interfacce, tali registri servono per inoltrare transazioni attraverso il bridge. Nell'interfaccia primaria, il 21554 risponde solo a quelle transazioni i cui indirizzi ricadono in uno dei suoi BAR. Tutte le altre transazioni sul bus primario vengono ignorate. Specularmente ciò accade anche nell'interfaccia secondaria.
Il bridge 21554 normalmente utilizza la decodifica diretta degli indirizzi quando inoltra transazioni da un'interfaccia all'altra.
Un indirizzo di memoria può essere pensato come un indirizzo base con un offset. Quando una transazione viene spedita downstream (cioè dal bus primario a quello secondario), l'indirizzo di bus primario viene mappato su un altro indirizzo del bus secondario sostituendo la base dell'indirizzo con un nuovo indirizzo di base, chiamato anche *indirizzo base tradotto*. Questo nuovo indirizzo base fa riferimento ad una nuova locazione nello spazio indirizzi del bus secondario. L'offset, invece, non viene modificato nella traduzione. La procedura è simile per transazioni upstream (cioè dal bus secondario a quello primario).
Ogni BAR ha il suo indirizzo base tradotto e sono programmabili tramite dei registri che sono mappati nello spazio di configurazione del dispositivo.

3. Implementazione della DSM

Le funzionalità della memoria condivisa sono implementate in un modulo del kernel di Linux che mette a disposizione una serie di device a caratteri. Ogni device rappresenta un segmento di memoria fisica di un nodo, resa accessibile a tutti gli altri. Inoltre, vengono implementate le primitive di sincronizzazione necessarie per controllare l'accesso alla memoria condivisa.

3.1. Il driver di memoria condivisa distribuita

Il driver consiste di una parte che si occupa dell'interfacciamento con il kernel, parte indipendente dall'harwdare, e di un'altra che si occupa di programmare e interloquire direttamente con il bridge.

Il ciclo di vita del driver può essere suddiviso in quattro fasi: *inizializzazione*, *inserimento di un nuovo nodo*, *lettura e scrittura su memoria condivisa* ed infine *rimozione di un nodo*. Queste fasi sono ben distinte, e spesso hanno tipologie di comportamento diverse a seconda che il driver sia in esecuzione sul System Master o su un Peripheral Master.

Inizializzazione: in questa fase si inizializza la memoria che si vuole effettivamente condividere (Figura 3). In tale memoria si mantiene anche la mappa del sistema, una struttura dati che tiene traccia dello stato di ogni nodo e contiene informazioni relative al buffer di memoria condivisa di ognuno di essi.

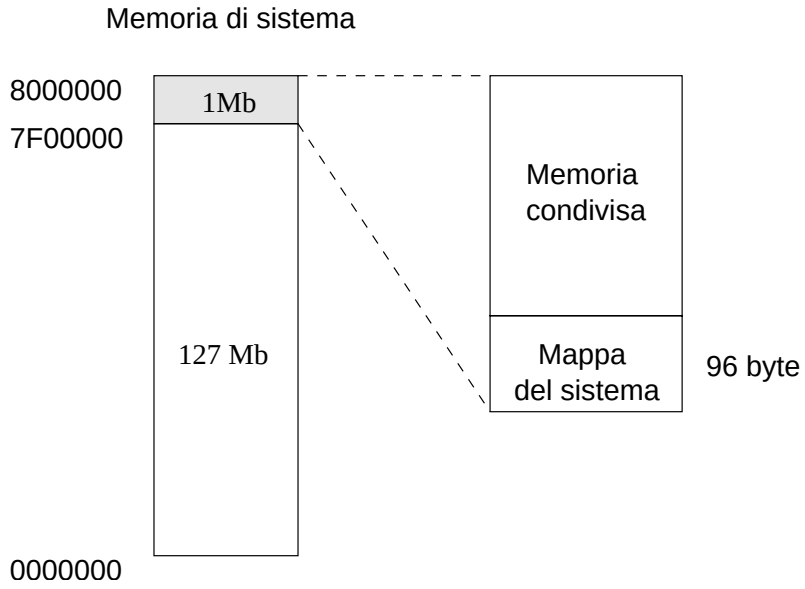

Figura 3: Segmento di memoria riservata alla condivisione via CompactPCI.

Viene quindi esaminato il bus PCI alla ricerca del bridge Intel 21554. Se ci troviamo nel System Master vengono trovati tanti bridge quanti sono i Peripheral Master presenti nel sistema. Ognuno di essi richiede 1 Mb di spazio di indirizzamento che serve per poter accedere al buffer situato dall'altra parte del bridge. Nel caso invece del Peripheral Master viene rilevato un unico bridge. La sua richiesta di memoria in questo caso è di 8 Mb, 1 Mb per ogni nodo.

Inserimento di un nuovo nodo: questa fase è sincronizzata mediante la generazione di determinati interrupt, ed aggiunge un nuovo nodo nel sistema notificando tale evento a tutti gli altri nodi già presenti.
La fase di inserimento ha inizio quando il modulo è stato caricato su entrambi i fronti del bridge, e quindi il System Master riceve un interrupt dal Peripheral Master appena inserito nel sistema. Sapendo qual'è il dispositivo che ha generato tale interrupt, il System Master è in grado di enumerarli e assegna un numero di nodo univoco che lo identifica all'interno del sistema.
La mappa del sistema viene quindi modificata marcando come "attivo" il nodo che ha generato l'interrupt. La modifica di tale mappa viene poi notificata ad ogni Peripheral Master. Il nuovo nodo è stato inserito ed è ora riconosciuto

all’interno del sistema. È quindi possibile, da ogni altro nodo, poter raggiungere il buffer da 1 Mb che esso ha messo in condivisione.

Lettura e scrittura su memoria condivisa: il System Master ha mappate nel suo spazio di indirizzamento tante zone di memoria quanti sono i Peripheral Master presenti nel sistema. Quando si effettuano degli accessi a queste zone di memoria si accede al buffer condiviso del relativo Peripheral (Figura 4(a)).

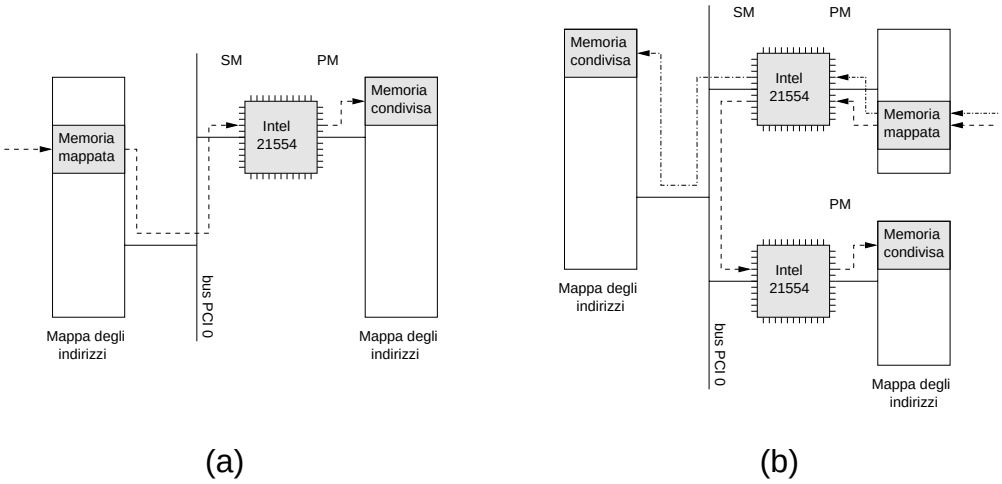

Figura 4: Accesso alla memoria condivisa da parte del System Master (a), e del Peripheral Master (b).

D’altra parte ogni Peripheral Master ha mappato, durante l’inizializzazione, una zona di memoria di 8 Mb, che viene interpretata come un array con 8 elementi da 1 Mb ciascuno. Ogni blocco da 1 Mb consente l’attraversamento del bridge e l’accesso allo spazio di indirizzamento del System Master. Da qui in poi si può scrivere nella memoria condivisa del System Master o in quella di un altro Peripheral Master (Figura 4(b)).

Rimozione di un nodo: la disinstallazione del driver porta alla rimozione di un Peripheral del sistema oppure, se il nodo rimosso è il System Master, alla cessazione della condivisione della memoria distribuita.
Se il nodo rimosso è un Peripheral, il System Master marca il nodo in questione come inattivo, quindi ridistribuisce la mappa del sistema aggiornata a tutti gli altri nodi attivi. Se il nodo rimosso è il System Master, viene inviato un interrupt ad ogni Peripheral attivo che, di conseguenza, rende invalido qualsiasi tentativo di utilizzo della memoria condivisa distribuita.

3.2. Primitive di sincronizzazione

Il problema della mutua esclusione attraverso memoria condivisa, nonostante risalga agli anni 70, è diventato pressante solo recentemente in seguito all’interesse da parte del mercato in soluzioni che permettano di connettere assieme hardware a basso costo con le minori modifiche possibili e dove quindi non è possibile disporre di primitive hardware per la sincronizzazione. Questa è la situazione del CompactPCI; quindi, ci si è focalizzati su algoritmi di sincronizzazione puramente software, senza l’ausilio di funzionalità hardware.

Il "bakery algorithm" eseguito dal nodo *i*

1. $getting_i = true$
2. $label_i = [\mathbf{max}\ \{label_j,\ \forall j = 1 \ldots N\}] + 1$
3. $getting_i = false$
4. ***for*** $j = 1$ ***to*** N ***do***
5. ***if*** $j \neq i$ ***then***
6. ***wait until*** $getting_i = false$
7. ***wait until*** $[(label_j = 0) \vee (label_i < label_j)]$
8. *CRITICAL SECTION*
9. $label_i = 0$

Figura 5: L'algoritmo *Bakery* di Lamport.

Sono stati considerati in particolare due algoritmi: il *Bakery Algorithm* di Lesile Lamport [6], e il *Bakery Bounded Algorithm* di Uri Abraham [1]. Il pseudocodice per questi due algoritmi è riportato in Figura 5 e 6 rispettivamente.
La complessità computazione dell'algoritmo di Lamport è *O(N)*, con *N* numero dei nodi. Ad oggi non esistono algoritmi migliori per risolvere il problema della mutua esclusione mediante modelli che fanno uso della memoria condivisa ed in assenza di operazioni atomiche.
Il maggiore difetto di questo algoritmo è che esso richiede l'uso di una variabile che possa incrementare il suo valore illimitatamente (come si può vedere dal comando 2 del pseudocodice in Figura 5). Dato che questo non è implementabile facilmente, esso rappresenta un limite pratico importante. Come soluzione a questo problema, è stato preso in considerazione l'algoritmo *Bakery Bounded* di Uri Abraham (Figura 6), che utilizza solo variabili limitate.

4. Valutazioni delle prestazioni

In questa sezione diamo una prima valutazione delle prestazioni che si possono ottenere attraverso l'uso di una DSM, in luogo dello scambio di messaggi basato sul protocollo di rete TCP/IP.
In tutti questi test, il sistema di riferimento consiste in una scheda Eurotech CPU-7630 e una scheda CPU-7631 [4]; la prima agisce come System device e la seconda come Peripheral sul backplane CompactPCI. Entrambe le schede sono dotate di un processore Pentium III a 500 MHz, e di 128MB di SDRAM PC133.
Trasferimento "raw": un primo confronto consiste nella misura approssimativa della banda massima raggiungibile durante un trasferimento di dati da nodo a nodo. A tale scopo si è eseguito un test che consiste nel trasferimento di 80 MB di dati con due modi diversi. Nel primo caso si utilizza il driver di MontaVista Software Inc., che implementa lo stack TCP/IP sul bus PCI (Figura 1(b)), e si copia un file da 80 MB con il comando *wget*.

Il "bakery bounded algorithm" eseguito dal nodo *i*

L'inizializzazione dei registri condivisi:

1. *v.evidence = *, v.status = 0, v.phase = 0*
2. ***write**(R_i,v)*

L'algoritmo:

1. *v.evidence = *, v.status = 1, v.phase = (v.phase + 1) mod 3*
2. ***write**(R_i,v)*
3. ***for** j = 1 **to** i − 1 do*
    a. ***read**(R_j, r_j)*
    b. *$v.evidence_j$ = r_j.phase*
4. *$v.evidence_i$ = v.phase*
5. ***write**(R_i,v)*
6. ***forall** j ≠ i*
    a. ***repeat read**(R_j, r_j)*
    b. ***until** (r_j.status = 0 ∨ r_j.evidence **dominates** v.evidence)*
7. *CRITICAL SECTION*
8. *v.status = 0*
9. ***write**(R_i,v)*

Figura 6: L'algoritmo *Backery Bounded* di Abraham.

Nel secondo caso invece si usa il driver della memoria condivisa distribuita descritto in questo lavoro. Siccome la dimensione di tale memoria è di 1MB, si copia il contenuto della memoria del nodo in cui si effettuano i test in quella di un nodo remoto per 80 volte consecutive. I dati rilevati sono riportati nella seconda riga della Tabella 1; come si vede, il trasferimento via DSM è quasi sei volte più veloce del trasferimento via driver di rete.

	DSM e sincronizzazione		TCP/IP via	
	Bakery	Bakery Bounded	NIC	CompactPCI
Synchronization	37 ± 5µs	56 ± 5µs	85 ± 5µs	27 ± 5µs
Trasferimento "raw" 80Mb	2.26 sec = 35.4 Mb/sec		- -	13.36 sec = 5.99 Mb/sec

Tabella 1: Prestazioni dei diversi modelli di comunicazione.

Sincronizzazione: oltre al trasferimento "raw" di dati, in un contesto generale è importante tenere conto anche dei costi di sincronizzazione. Un semplice

benchmark, che verrà chiamato *Synchronization*, è costituito da una coppia di programmi che si sincronizzano a vicenda consecutivamente, come descritto dal seguente pseudocodice:

Il programma "first"

1. **down**(*sem1*)
2. **up**(*sem2*)

Il programma "second"

1. **up**(*sem1*)
2. **down**(*sem2*)

Come si vede, questo programma esegue solamente le operazioni necessarie alla mutua esclusione e per questo motivo è il più adatto per valutare le prestazioni degli algoritmi implementati.
Questo pseudocodice è stato implementato sia direttamente sulla DSM usando le primitive di sincronizzazione proposte, sia in una versione *client/server* basata sul stack TCP/IP. Quest'ultima versione è stata testata su una interfaccia di rete virtuale che simula il protocollo TCP/IP sulla DSM secondo lo schema della Figura 1(b), usando il driver della MontaVista già citato. Per completezza si è eseguito questo test anche usando TCP/IP su Fast Ethernet, secondo il modello di comunicazione di Figura 1(a). I risultati ottenuti sono riportati nella prima riga della Tabella 1. Come si può vedere, l'algoritmo che ottiene le migliori prestazioni è sicuramente il *Bakery* di Leslie Lamport, che però presenta il problema dell'uso di variabili illimitate. Segue il *Bakery Bounded* di Uri Abraham, che non necessita di varabili illimitate. Rispetto a quest'ultima soluzione, la sincronizzazione via Ethernet è più lenta di circa il 50%. Invece, la soluzione che si basa sul protocollo TCP/IP implementato sulla DSM risulta più lento di ben tre ordini di grandezza.

Problemi e limitazioni hardware: nel corso di questa ricerca è stato verificato che l'architettura proposta è affetta da alcuni problemi hardware di *bus contention* sul backplane CompactPCI; ad esempio alcune transizioni su tale bus possono fallire ed inoltre il trasferimento dei dati in modalità "64 bit" si attiva solo lungo una direzione. Quest'ultimo fatto porta a risultati diversi a seconda di come venga effettuato l'assegnamento di un valore ad una variabile (remota); nei casi in cui i programmi di valutazione non siano simmetrici ed il carico della comunicazione non è bilanciato nelle due direzioni, i tempi risultanti cambiano a seconda della direzione della comunicazione.
La dipendenza dei risultati dalla direzione dell'assegnamento è causa dell'elevato valore dell'errore assoluto nelle misure riportate. Le stime riportate in tabella rappresentano la media dei risultati ottenuti per ogni assegnamento possibile; date le problematiche sopra esposte i risultati sono soggetti a forte varianza e di conseguenza l'errore assoluto che se ne ricava è elevato.

5. Conclusioni

In questo lavoro è stata descritta una implementazione di una *memoria distribuita condivisa* su un cluster di calcolatori collegati da un backplane PCI, in luogo di una infrastruttura di rete a pacchetti.
L'architettura data è di tipo multicomputer, ed ogni bus CompactPCI può scalare facilmente fino a 8 nodi; in linea di principio, ogni nodo può essere un calcolatore SMP, raggiungendo così 16-32 processori complessivamente. Si

può espandere facilmente questa architettura con dei bridge PCI-PCI, che si comportano come dei “super buffer” e consentono di mettere in comunicazione tra di loro diversi bus. Infatti gli interrupt, le informazioni plug-and-play e i dati vengono trasferiti automaticamente attraverso questo bridge. Un ulteriore vantaggio è che ogni singolo bus CompactPCI appartenente al sistema può effettuare trasferimenti di dati sui suoi nodi in maniera indipendente.
Quindi, l’architettura proposta è più facile da costruire, e con costi minori, rispetto ad una architettura multiprocessore ad essa paragonabile. Inoltre, il driver implementato e qui descritto permette di utilizzare il modello della memoria condivisa distribuita, più intuitivo del modello a scambio di messaggi dal punto di vista della programmazione, con buoni risultati.
A conclusione si può dire che la ricerca svolta ha mostrato che è conveniente utilizzare la memoria condivisa distribuita in sostituzione dei classici protocolli di rete, soprattutto perché vi sono ancora molte migliorie da potersi effettuare sia sull’hardware della macchina in questione sia sul codice che è stato scritto.
Le chiamate di sistema che permettono di gestire le interfacce Ethernet, e tutte le connessioni su protocollo TCP/IP, sono incluse nel kernel Linux fin dalle sue prime versioni ed hanno quindi raggiunto un elevato grado di ottimizzazione. Per contro il meccanismo utilizzato per permettere all’utente di chiamare gli algoritmi di mutua esclusione in *user space* non è dei migliori, esso, infatti, ricorre alla chiamata di sistema IOCTL all’interno della quale è presente un costrutto di tipo *case* che permette di selezionare l’algoritmo e le funzioni di mutua esclusione desiderate ma all’aumentare di queste ultime la soluzione diventa costosa ed inefficiente. Una possibile soluzione potrebbe essere quella di permettere la chiamata da *user space* di uno solo degli algoritmi, quello che meglio si adatta alle proprie esigenze, e di permetterne l’accesso tramite chiamate di sistema che già di per sé specificano la funzione desiderata, come ad esempio possono essere la SEMOP, SEMGET e SEMCTL.
Altre possibili migliorie sono:

- Ottimizzare l’implementazione dell’algoritmo del *Bakery Bounded* di Uri Abraham per rendere le sue prestazioni il più vicine possibile a quelle del *Bakery* di Leslie Lamport;
- Rendere dinamiche le strutture dati dell’algoritmo;
- Automatizzare l’assegnamento dell’id di processo.

Ringraziamenti Gli autori ringraziano la Eurotech S.p.A. e la Ascensit S.p.A. di Amaro (Udine) per il supporto hardware e software offerto durante lo sviluppo della presente ricerca. CompactPCI è un marchio registrato del PCI Industrial Computer Manufacturers Group (PICMG).

6. Riferimenti bibliografici

[1] U. Abraham, December 1995, *Bakery Algorithms - Technical report*, Department of Methematics, Ben Gurion University, Beer-Sheva, Israel.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. N. Seizovic and W.K. Su, 1995, *Myrinet: A gigabit-per-second local area network*, IEEE Micro, 15(1):29–36.

[3] G. Burns, R. Daoud and J. Vaigl, 1994, *LAM: An Open Cluster Environment for MPI*, In Proceedings of Supercomputing Symposium, pages 379–386.

[4] EuroTech, Via Jacopo Linussio, n.1. Amaro (UD) ITALY, Jan. 2001, *Eurotech CPU-630/7631 CompactPCI 6U SBC User's Manual*, 1st edition.

[5] Intel, September 1998, *21554 PCI-to-PCI Bridge for Embedded Applications*, Intel.

[6] L. Lamport, February 1987, *A Fast Mutual Exclusion Algorithm*, ACM Transactions on Computer Systems, 5(1):1–11.

[7] Montavista linux professional edition, 2001.

[8] PCI Industrial and Computer Manufacturers Group (PICMG), Wakefield, Mass, 1995, *CompactPCI Specification*, http://www.picmg.com.

[9] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra and S. Huss-Lederman, 1995, *MPI: The Complete Reference*, MIT Press, Cambridge, MA, USA.

[10] J. M. Squyres and A. Lumsdaine, September/October 2003, *A Component Architecture for LAM/MPI*, In Proceedings 10th European PVM/MPI Users' Group Meeting, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, Springer-Verlag.

[11] A. S. Tanenbaum, 2000, *Architettura dei Computer*, Prentice-Hall.

[12] A. S. Tanenbaum and M. van Steen, 2002, *Distributed Systems - Principles and Paradigms*, Prentice-Hall.

[13] M. S. Warren, D. J. Becker, M. P. Goda, J. K. Salmon and T. L. Sterling, 1997, *Parallel supercomputing with commodity components*, In H. R. Arabnia, editor PDPTA, pages 1372–1381, CSREA Press.

A Accesso alle funzionalità della DSM

In questa sezione si descrivono brevemente le API per l'accesso alle funzionalità della DSM descritta in questo lavoro.
Le memorie condivise di ogni nodo sono rappresentate dai file speciali (device di tipo carattere) */dev/shm_cnet0* a */dev/shm_cnet7*, con l'ultima cifra ad identificare il numero del nodo.
Questi device possono essere acceduti come se fossero uno stream di byte (come un file). Precisamente, sono stati implementati l'I/O tradizionale (i.e., con chiamate "read" e "write"), ed il *memory-mapped* I/O.
Ogni device può essere letto come un qualsiasi file (p.e., con *fopen*); è poi possibile effettuare su di esso le normali operazioni di read e write. Ad esempio con il comando *cat /dev/shm_cnet0 > /dev/shm_cnet1* si trasferisce tutto il contenuto della memoria condivisa distribuita del nodo 0 al nodo 1.
Inoltre, è spesso vantaggioso usare la funzionalità *mmap* poiché permette di mappare il contenuto di un file (in questo caso, una memoria remota) in una sezione dello spazio di indirizzi del processo, facilitando la stesura di programmi che si basano sulla condivisione di dati (ad esempio, thread).
Con una semplice chiamata del tipo

```
fd = fopen("/dev/shm_cnet1","rw",0);
mmap(NULL, shared_memory_size, PROT_READ |
        PROT_EXEC | PROT_WRITE, MAP_SHARED, fd, 0);
```

tutte le variabili dichiarate nello scope del processo saranno allocate in memoria condivisa, in particolare nella memoria del processore specificato tramite il file descriptor *fd* (il nodo 1, in questo caso).

Infine, le primitive di mutua esclusione sono state rese accessibili a livello utente tramite la chiamata di sistema *ioctl*. La chiamate in generale è *ioctl(fd, c, proc)*, ove *fd* è un device di memoria condivisa, *proc* un numero univoco rappresentante il processo, e *c* un codice di controllo. I codici di controllo principali sono i seguenti:

- *IOCTL_BAKERY_INIT*: inizializza il mutex sulla memoria condivisa.
- *IOCTL_BAKERY_ADD_NODE:* aggiunge il processo *proc* all'uso del mutex sulla memoria condivisa *fd*.
- *IOCTL_BAKERY_REMOVE_NODE*: rimuove il processo *proc* all'uso del mutex sulla memoria condivisa *fd*.
- *IOCTL_BAKERY_MUTEX_DOWN:* down (bloccante) sul mutex per il processo *proc.*
- *,IOCTL_BAKERY_MUTEX_UP*: l'up sul mutex per il processo *proc*.