Università degli Studi di Padova

Facoltá di Ingegneria

Corso di Laurea in Ingegneria Informatica

Stato dell'arte dei processori multicore

Laureando

Daniele Bisello

Relatore

Prof. Sergio Congiu

Anno Accademico 2011/2012

Indice

Capitolo 1

Introduzione ai processori multicore

1.1 Introduzione

Per diversi decenni la progettazione dei processori è andata verso un aumento delle prestazioni del singolo processore, cercando di aumentare il più possibile la frequenza di clock e quindi il numero di operazioni eseguite nell'unità di tempo. Purtroppo questa soluzione si è rivelata molto dispendiosa in termini di potenza assorbita dalla circuiteria, ed essendo i processori largamente utilizzati in sistemi embedded, nei quali l'energia (batteria) è una risorsa di importanza vitale, la soluzione alternativa, è stata quella di far lavorare assieme più processori (magari meno potenti, e quindi meno dispendiosi in termini di energia) raggiungendo una potenza di calcolo anche maggiore di quella ottenuta con un singolo processore. Per questo motivo sono stati inseriti in un singolo chip più processori, creando architetture chiamate microprocessori multicore (o microprocessori multiprocessore). Ovviamente far lavorare molti processori assieme è molto più complesso di farne lavorare uno solo, perché bisognerà sicuramente far si che questi si accordino sull'utilizzo dell'eventuale memoria comune, sui compiti che ognuno di essi dovrà compiere e banalmente si dovranno ottenere prestazioni maggiori, altrimenti non sarebbe giustificato lo sforzo. L'utilizzo di più processori porta anche a benefici non indifferenti, ad esempio in caso di guasto di un processore, si può far si che venga garantita in ogni caso l'esecuzione di un programma con un processore in meno. Un altro vantaggio è indubbiamente l'aumento delle prestazioni, in quanto si possono eseguire su processori diversi delle attività tra loro indipendenti.

1.2 Condivisione della memoria

Il problema della condivisione della memoria tra i core si può affrontare con due approcci:

1. avere un unico spazio di indirizzamento per tutti i core, (SMP Shared Memory Multiprocessor). I processori comunicano attraverso variabili condivise contenute nella memoria. Inoltre i processori che usano questo metodo di indirizzamento della memoria sono suddivisi in multiprocessori con memoria ad accesso uniforme (UMA, Uniform Memory Access) ovvero impiegano lo stesso tempo per accedere alla memoria principale indipendentemente dalla parola richiesta e dal processore che ha effettuato la richiesta e multiprocessori ad accesso non uniforme (NUMA, Nonuniform Memory Access) ovvero alcuni accessi alla memoria possono essere più rapidi di altri

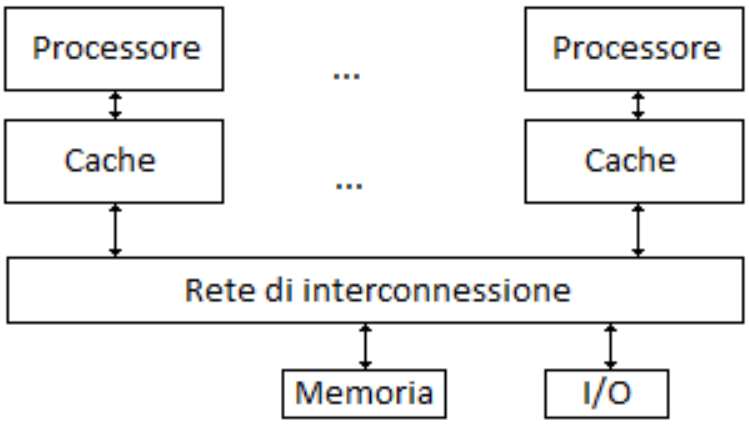

Figura 1.1: memoria condivisa

2. ciascun processore abbia il proprio spazio privato di indirizzamento fisico. La comunicazione tra i processori avviene quindi per mezzo di un scambio di messaggi di tipo send e receive. I cluster sono costituiti da un insieme di calcolatori di base connessi tra loro attraverso connessioni di I/O. Esistono però alcuni inconvenienti, come il costo della manutenzione (sono presenti più calcolatori), la lentezza dei mezzi usati per lo scambio dei messaggi e il sovraccarico dovuto alla suddivisione della memoria (ad es. N macchine, N sistemi operativi installati)

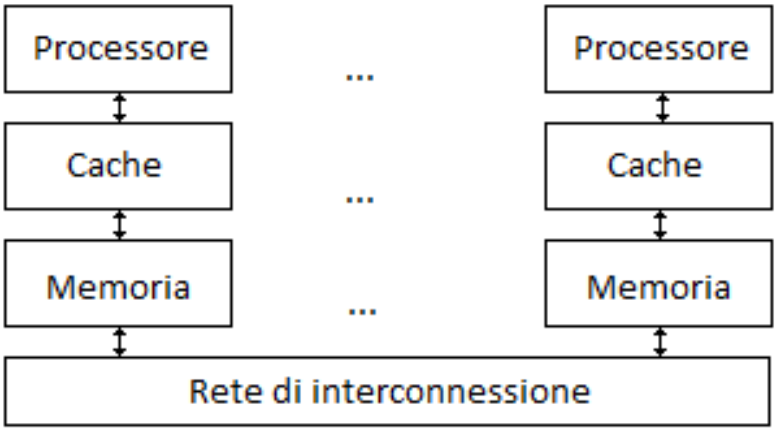

Figura 1.2: scambio messaggi

1.3 Multithreading hardware

Il multithreading hardware permette a più thread di condividere le unità funzionali dello stesso processore mediante una sovrapposizione. Chiaramente il processore deve poter duplicare lo stato di ciascun thread, e l'harware deve essere capace di supportare i rapidi passaggi dall'esecuzione di un thread all'altro. Esistono tre approcci al multithreading:

1. il multithreading a grana fine, l'idea che sta alla base di questo approccio è che ad ogni istruzione si passa da un thread all'altro usando una modalità round robin.

2. il multithreading a grana grossa, nel quale il thread in esecuzione cambia solo quando si presenta uno stallo "costoso", come un miss nella cache di secondo livello.

3. il multithreading simultaneo (SMT, Simultanuous MultiThreading). Questo tipo di multithreading si basa sul fatto che i processori moderni sono dotati di parallelismo dell'esecuzione e quindi hanno più unità funzionali che lavorano in parallelo di quante ne possa usare un singolo thread.

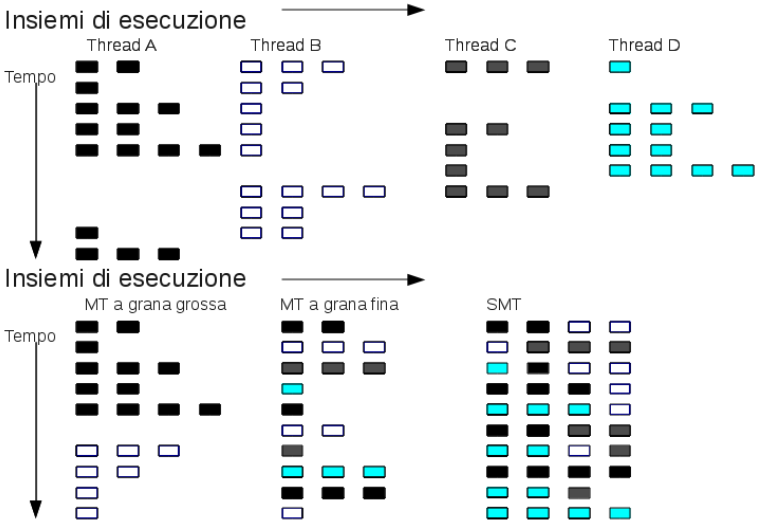

Figura 1.3: grana thread

1.4 Classificazione hardware parallelo

Un tipo di classificazione dell'hardware parallelo proposto negli anni sessanta è quello sul numero dei flussi di istruzioni e dati. Esistono quattro categorie:

- MIMD (Multiple Instruction streams, Multiple Data Streams)più flussi di istruzioni e più flussi di dati. Ad esempio un grosso problema può essere scomposto in più programmi che vengono eseguiti su più processori.
- SISD (Single Instruction stream, Single Data stream)un unico flusso di istruzioni e un unico flusso di dati. Ad esempio i tradizionali computer monoprocessore.
- MISD (Multiple Instruction stream, Single Data stream)flussi multipli di istruzioni e flussi singoli di dati. Non ci sono esempi commerciali.
- SIMD (Single Instruction stream, Multiple Data stream)la singola istruzione viene applicata a diversi flussi di dati (calcolo vettoriale)

Sui calcolatori MIMD è possibile usare una tecnica di programmazione SPMD (Single Program Multiple Data)la quale consiste nel mandare un flusso di istruzioni su più processori. Ad esempio un programma che viene eseguito su più processori. Di fatto per il programmatore è come se fosse un SISD [1]

Capitolo 2

Tiled Multicore Processors

2.1 Introduzione

In questo capitolo si descrive in modo basilare una nuova architettura di processori che permette di sfruttare al meglio le nuove tecnologie grazie alle quali è possibile inserire sempre più transistor nello stesso chip. Questa nuova architettura si chiama tiled architecture (architettura a piastrella) e consiste nell'inserire nello stesso chip un processore e un elemento di collegamento (switch o router), creando quindi una piastrella facilmente collegabile ad altre. Questo insieme di piastrelle permette perciò di incrementare notevolmente le prestazioni complessive del sistema, sebbene prese singolarmente non siano molto potenti.

2.2 La fine dei processori monolitici?

Nel corso degli anni la legge di Moore è sempre stata verificata, infatti ogni anno il numero di transistor che sono stati inseriti in un singolo chip è sempre aumentato. Purtroppo però si è dimostrato con i fatti che non basta aumentare la potenza dell'hardware per avere prestazioni maggiori. Negli ultimi anni infatti sono stati creati i processori super-scalari, questi processori hanno delle prestazioni elevatissime grazie alla notevole frequenza di clock e all'architettura estremamente raffinata e complessa, ma d'altro canto consumano anche molta energia. Inoltre l'elevata frequenza di lavoro limita anche la scalabilità del sistema, in quanto maggiore è la frequenza di clock , minore è l'area del chip raggiungibile dal segnale. Per poter comunque aumentare le prestazioni dei calcolatori, ridurre l'assorbimento di potenza e far aumentare la scalabilità, si è pensato di far lavorare assieme più processori, anche se con prestazioni non confrontabili singolarmente con i processori super-scalari.

2.3 Tiled Multicore Architectures

La nuova architettura a piastrelle evita quindi l'inefficienza dei processori monolitici e aumenta la scalabilità del sistema. I tiled multicore contengono al loro interno una certo numero di core invece di uno molto grande, questo perchè sono più efficienti sia sotto l'aspetto del consumo di energia, che sotto l'aspetto di consumo di area utile del chip. In assenza di colli di bottiglia, il limite di potenza di calcolo raggiungibile con questa architettura è dovuto al numero di core che lavorano insieme, e non la frequenza di clock raggiunta dal singolo. Il concetto chiave di questa architettura è come sono connessi tra di loro questi processori. Una piastrella è composta da un core, più un elemento di connessione con gli altri core, ovvero uno switch. Collegando le varie piastrelle mediante questo switch, otteniamo una rete di comunicazione su un chip. Tutte le comunicazioni tra i core e con l'esterno avvengo quindi tramite questa rete. I vantaggi nell'adozione di questa soluzione sono molti, cominciando dalla maggiore scalabilità, infatti per aumentare le prestazioni basta aggiungere piastrelle e non riprogettare tutto, fino alla maggiore versatilità rispetto ai general-pupose tradizionali, in quanto forniscono molte più risorse, che sta poi all'applicazione decidere a chi e cosa assegnare, nel modo più efficiente possibile.

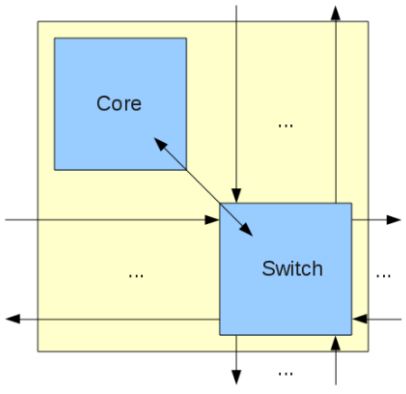

Figura 2.1: esempio piastrella

2.4 Tilera®

Un'azienda molto importante nella progettazione e nella costruzione e vendita di questa nuova tipologia di processori è indubbiamente Tilera®. Il suo prodotto di punta è TILE64™. Questo chip contiene al suo interno 64 processori uguali, interconnessi con una rete su chip proprietaria, la Tilera's iMesh™. Ognuna di queste piastrelle comprende il processore, le memorie cache L1 e L2 e lo switch per la connessione con la rete. [2] [3] [4] [5] [6]

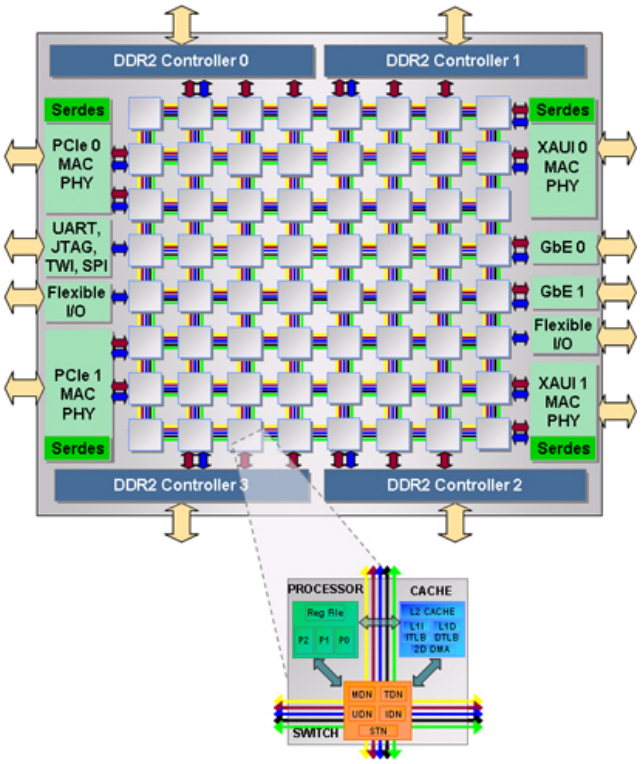

Figura 2.2: Tile64

Capitolo 3

On-chip networks for multicore systems

3.1 Introduzione

La combinazione tra il sempre crescente consumo di energia e le sempre minori performance dei sistemi monoprocessore, ha decretato l'avvento dei processori multicore. L'enorme incremento del numero di transistor che è possibile inserire in un chip è gestibile molto meglio con questa architettura, piuttosto che con il vecchio metodo di progettare un singolo processore molto potente. Negli ultimi anni tutte le aziende produttrici di processori hanno immesso nel mercato chip con sempre più core al loro interno. Questo aumento di core che comunicano all'interno di uno stesso chip ha fatto aumentare notevolmente l'importanza del metodo usato per la comunicazione stessa. È per questo motivo che il modello di reti a pacchetto sta sostituendo i tradizionali metodi di comunicazione all'interno dello stesso chip, come ad esempio i bus. La progettazione delle nuove reti sul chip, possono ispirarsi alle reti di calcolatori tradizionali, stando attenti però al fatto che le specifiche del progetto sono molto diverse. Le reti sul un chip devono avere bassissima latenza, banda molto elevata e inoltre non devono assolutamente sprecare energia o spazio.

3.2 Fondamenti

Una rete su un chip è definita da diversi parametri: la sua topologia, gli algoritmi di routing, il controllo di flusso e la micro-architettura dei router. La topologia della rete definisce come i nodi sono interconnessi tra loro e quindi determina il numero di percorsi che un messaggio può percorrere per

arrivare al destinatario. Gli algoritmi di routing invece si occupano di scegliere un percorso specifico tra tutti quelli disponibili, possibilmente quello migliore. Il controllo di flusso specifica di fatto come un messaggio attraversa un percorso scelto e anche l'inoltro e la bufferizzazione dello stesso nei router. Infine la micro-architettura dei router implementa su hardware sia gli algoritmi di routing, che il controllo di flusso. Questi quattro parametri influiscono molto sia nel costo che nelle prestazioni finali della rete e quindi bisogna operare delle scelte molto oculate. Due parametri da tenere molto in considerazione in fase di progetto sono la latenza e la banda che si vuole ottenere, in genere una rete su un chip deve avere bassa latenza e alta banda. Un altro parametro molto importante è il consumo di energia, che dipende in parte dall'hardware ma anche dal traffico che si dovrà gestire.

3.3 Topologia

Gli effetti della topologia della rete sono molto profondi, in base a questa scelta un messaggio effettuerà un diverso numero di salti per raggiungere la destinazione, influenzando molto pesantemente la latenza. La topologia della rete influenza molto anche il consumo di energia, infatti più nodi verranno percorsi, più energia viene usata per far arrivare a destinazione il messaggio. La topologia della rete influenza anche l'affidabilità della rete, infatti essa determina il numero di percorsi alternativi che un messaggio può percorrere in caso di rottura di qualche collegamento. Il costo di implementazione complessivo della topologia è dovuto al grado di ciscun nodo e dal costo di realizzazione della rete stessa sul chip. Le tre topologie più diffuse sono l'anello, la maglia e il toro.

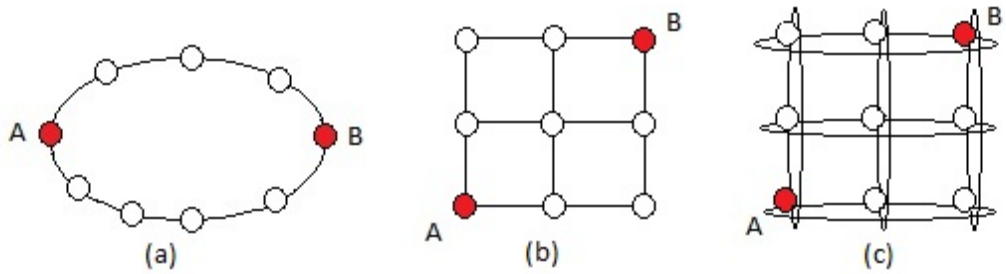

Figura 3.1: Topologie

Le tre topologie hanno caratteristiche molto diverse, infatti con lo stesso numero di nodi, il percorso da un nodo all'altro ha più salti nell'anello e

nella maglia che non nel toro, mentre per quanto riguarda il numero di percorsi possibili, è sempre il toro ad averne un numero maggiore, e quindi una tolleranza al guasto maggiore. La topologia ad anello anche se è quella con performance più scadenti, ha la caratteristica di essere quella più economica da costruire se paragonata alle altre due perchè ha tutti i nodi con grado due mentre le altre due hanno grado quattro (nell'esempio in figura). La topologia ideale non è determinabile a priori, infatti la scelta va fatta dopo una attenta analisi di diversi fattori come la latenza o la banda che si vuole ottenere, e anche in base al budget disponibile e all'utilizzo che poi si farà del sistema.

3.4 Algoritmi di routing

L'obiettivo degli algoritmi di routing è quello di distribuire il traffico in modo uniforme tra tutti i percorsi disponibili così da evitare congestioni dovute al traffico e massimizzare il throughput. Un altro obiettivo è quello di aumentare la tolleranza ai guasti, infatti è compito degli algoritmi di routing aggirare i nodi non funzionanti e garantire in ogni situazione la comunicazione tra mittente e destinatario. Questi obiettivi devono essere raggiunti senza aumentare troppo l'utilizzo di risorse, infatti la circuiteria non deve occupare troppo spazio nello stampato. Per quanto riguarda il consumo di energia, questo è trascurabile per quanto riguarda la circuiteria, ma può incidere se i percorsi dei messaggi non sono ottimi. Esistono diversi algoritmi di routing, ma quello più usato è DOR (dimension-ordered routing) grazie alla sua semplicità. Con questo algoritmo un messaggio attraversa la rete dimensione per dimensione, fino a raggiungere le coordinate della dimensione nella quale si sta viaggiando, e poi si passa a viaggiare nella dimensione seguente e così via fino a raggiungere la destinazione. Ad esempio in una topologia a due dimensioni un messaggio prima viaggia nella direzione X, poi quando ha raggiunto la coordinata X della destinazione, comincia a muoversi nella direzione Y fino ad arrivare alla destinazione.

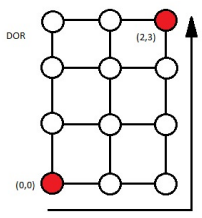

Figura 3.2: Esempio DOR

L'algoritmo DOR è un esempio di algoritmo deterministico, ovvero un messaggio che deve andare da A a B percorrerà sempre lo stesso percorso. Un'altra classe di algoritmi sono quelli ignari (oblivious) nei quali un messaggio scelglierà un percorso piuttosto che un altro senza curarsi della situazione della rete. Ad esempio un router sceglierà a caso il collegamento nel quale instraderà il messaggio. Un algoritmo più sofisticato è quello adattativo, nel quale un percorso viene scelto in base alla situazione della rete. Quando si sta scegliendo o creando un algoritmo di routing, non si deve prendere in considerazione solo il ritardo, l'energia e l'affidabilità, bisogna anche far si che non si presentano situazione di deadlock. Situazioni di questo tipo sono evitate imponendo agli algoritmi il divieto di creare cicli nei percorsi dei messaggi.

3.5 Controllo di flusso

Il controllo di flusso gestisce l'allocazione dei collegamenti e dei buffer della rete. Determinano quindi come quando i buffer e i collegamenti devono essere assegnati e a quale messaggio, la granularità e in che modo queste risorse vengono condivise tra i messaggi. Il controllo di flusso incide anche nella qualità del servizio della rete, infatti è lui che decide il tempo di partenza di un messaggio ad ogni salto. Tra i più importanti algoritmi di controllo del flusso ci sono lo store-and-forward, il virtual cut-through e il wormhole. Il controllo di flusso di tipo store-and-forward impone che ogni nodo debba aspettare l'arrivo dell'intero pacchetto prima di cominciare ad inoltrarlo sulla linea successiva. Questo comportamento implica grossi ritardi ad ogni nodo, e non è assolutamente consigliabile in una rete on-chip, in quanto il ritardo deve essere il minore possibile. Il controllo di flusso di tipo virtual cut-through invece permette l'inizio della trasmissione di un pacchetto prima del completamento della ricezione dello stesso, quindi il ritardo viene diminuito drasticamente, ma purtroppo le risorse non vengono assegnate nel modo migliore possibile, perchè ad esempio servono dei buffer molto capienti. Infine il wormhole flow control impone lo stesso comportamento di virtual cut-through ma in più permette di gestire al meglio l'allocazione delle risorse quali banda, buffer e collegamenti. Un altra tecnica per il controllo del flusso è il virtual-channel che consiste nel fare in modo che due pacchetti possano condividere uno stesso collegamento, in modo che se uno di questi è bloccato, si può comunque procedere con la trasmissione. [2] [3] [7]

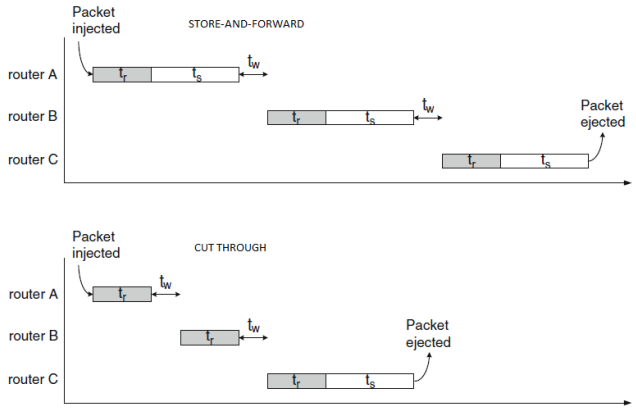

Figura 3.3: Controllo di flusso

Capitolo 4

Composable multicore chips

4.1 Introduzione

Due problemi da risolvere quando si sta progettando un chip multicore sono decidere quanti processori e di quale dimensione mettere in un chip e quanta area assegnare alle varie strutture che servono a sfruttare il parallelismo in ogni processore. Esistono diversi approcci per la soluzione di questi problemi, uno di questi è ad esempio inserire in un chip pochi processori ma molto potenti. Questa soluzione va bene per usi generici i quali hanno thread a grana grossa che sfruttano molto il parallelismo a livello di istruzione. Un approccio completamente diverso è invece usare un numero molto elevato di processori di potenza modesta. Questa soluzione va molto bene per applicazioni che richiedo un alto livello di parallelismo a livello di thread. Esistono comunque dei compromessi tra i due approcci che miscelano bene le caratteristiche di entrambe le soluzioni.

4.2 Processori Componibili

Questo approccio fornisce un sistema che si adatta al compito che gli viene assegnato in quanto riesce ad eseguire task grandi e piccoli in modo molto efficiente. L'idea di base è aggregare un grande numero di piccoli processori per formare un grande processore logico in grado di affrontare lavori più pesanti, che singolarmente questi piccoli processori non potrebbero portare a termine in tempo utile. Comporre in questo modo i processori rende il sistema adattabile al grado di parallelismo sia a livello di istruzioni che di thread in modo del tutto trasparente alle applicazioni software. I processori componibili hanno il vantaggio di poter essere configurati per le più svariate situazioni, infatti possono affrontare sia lavori destinati a processori composti

da pochi core molto potenti, sia quelli destinati a processori composti da svariati piccoli core. Per supportare questo grado di flessibilità, le varie componenti di questa architettura devono essere il più possibile distribuite in quanto se fossero centralizzate si avrebbe una grosso calo della scalabilità.

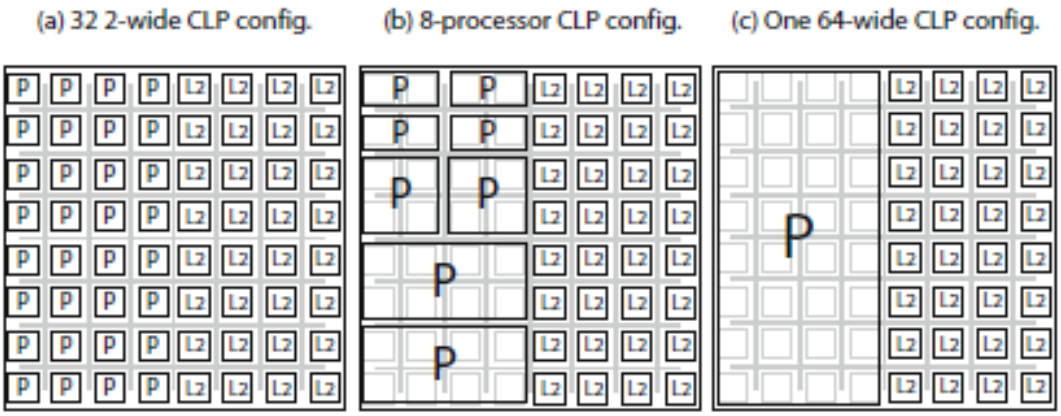

Figura 4.1: Processore logico

4.3 ISA

Progettare un processore componibile che riesca a scalare bene, usando i set di istruzioni tradizionali (RISC o CISC) si è rivelata una grossa sfida. In primo luogo le architetture convenzionali hanno un unico punto dove viene eseguito il fetch e il commit delle istruzioni, e quindi serve una centralizzazione di questa operazione, in modo da mantenere una esecuzione sequenziale delle istruzioni. In secondo luogo, i programmi compilati per architetture convenzionali, contengono di frequente salti di flusso nelle istruzioni, il che implica che servono dei rapidi cambiamenti di contesto. E terzo, l'aggiunta di più di una unità di elaborazione tipicamente richiede l'esecuzione di un broadcast degli output delle ALU, e l'hardware non può tracciare facilmente i produttori e i consumatori dei dati. Creare microarchitetture logiche più grandi, partendo da processori più piccoli è una delle principali sfide del progetto di processori componibili. Per affrontare il problema della scalabilità a livello strutturale, è stato introdotto un nuovo set di istruzioni, chiamate Explicit Data Graph Execution (EDGE). Una architettura di tipo EDGE supporta l'esecuzione di molti programmi, mappando i flussi di dati generati dal compilatore in un substrato distribuito di computazione, immagazinamento e controllo. Le due caratteristiche principali di EDGE sono

l'atomicità di esecuzione dei blocchi di istruzioni e la comunicazione dirette delle stesse all'interno di un blocco. Il TRIPS ISA è una istanza di EDGE ISA il quale aggrega 128 istruzioni in un unico blocco che viene eseguito in modo atomico. In questo modo ogni blocco di istruzioni viene eseguito come se fosse una singola istruzione. Inoltre questo modello riduce il numero di volte in cui bisogna eseguire una predizione di ciò che si vuole eseguire in seguito. La comunicazione diretta dei risultati di una istruzione appartenente ad un blocco con un'altra appartenente al blocco stesso, permette di non avere strutture centralizzate. La comunicazione viene implementata inserendo il destinatario del risultato di una istruzione, all'interno dell'istruzione stessa se il destinatario risiede anch'esso all'interno del blocco. In caso contrario il risultato viene condiviso mettendo il risultato dell'operazione all'interno di un registro. [2]

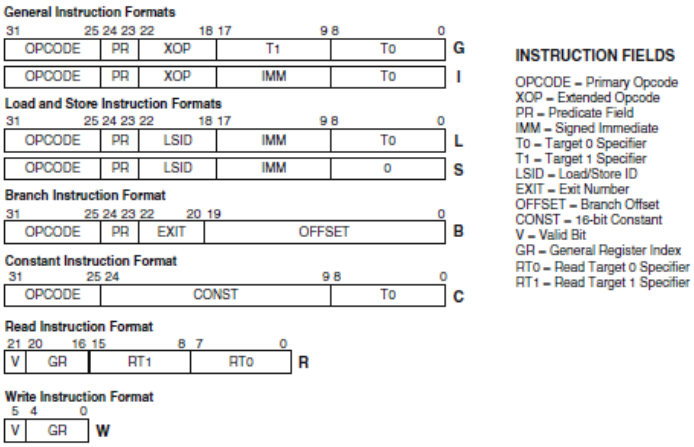

Figura 4.2: Formato istruzioni

Capitolo 5

Speculative multithreading architecture

5.1 Introduzione

I progettisti di architetture di processori sono alla continua ricerca di nuovi modi per rendere più produttivo il crescente numero di transistor disponibili dalle nuove generazioni di tecnologie di semiconduttori. I superprocessori moderni sfruttano al massimo i transistor per aumentare il livello di istruzioni eseguibili in parallelo. Questo approccio ha portato a due grossi problemi: i processori creano una sola finestra di istruzioni per cercare istruzioni indipendenti, e l'aumento di questa finestra non implica necessariamente un aumento delle performace. Inoltre molti circuiti chiave di questi processori superscalari consumano una gran quantità di potenza. Un modo per alleviare entrambi i problemi è quello di adottare i processori multicore, che risolvono il problema usando tanti piccoli processori che sono più efficienti di uno molto grande. Sorge però un nuovo problema, per aumentare il livello di parallelismo di questi piccoli processori, i programmatori sono obbligati a scrivere programmi paralleli. Questo tipo di programmazione sfortunatamente non è semplice, perchè introduce problemi non banali come la gestione dei dati (in modo distribuito) e la sincronizzazione. Il fardello della gestione dei dati è molto pesante in quanto è una questione molto complicata e che ha un grosso impatto sulle prestazioni del programmi, mentre la gestione della sincronizzazione può portare ai problemi classici di deadlock. Negli anni passati il problema della programmazione parallela è passato un po' in secondo piano perchè era un mercato di nicchia (come quello dei server database) e quindi questi problemi venivano affrontati per lo più da esperti nel settore. Ora che questi chip sono commercializzati anche al grande pubblico, si presuppone

che anche programmatori non esperti nel mondo dei programmi paralleli riescano a scrivere software performante per questi sistemi. Per questo sono stati introdotti gli speculatively multithread processors che partizionano un normale programma sequenziale in frammenti chiamati task che vengono poi eseguiti in parallelo sui vari core. Sebbene non sia garantito che i task siano indipendenti, si presume che l'esecuzione degli stessi sia indipendente. Questo approccio aiuta quindi l'inevitabile passaggio dall'era dei programmi di tipo sequenziale a quella di tipo parallelo.

5.2 Obiettivi

Questa architettura per essere competitiva deve soddisfare certi requisiti, ad esempio l'architettura dovrebbe impiegare hardware commerciale in modo che la velocità di clock non sia influenzata negativamente. Inoltre l'eventuale aumento del numero di core, non deve influenzare negativamente la frequenza di clock e questo è possibile solo se non vengono centralizzate le strutture hardware. Infine, tutti dovrebbero essere in grado di sfruttare il parallelismo in modo efficiente e senza sprechi di energia. Il software che insiste su questo hardware deve avere le seguenti caratteristiche:

- bisogna poter scrivere i programmi nei linguaggi ordinari
- l'hardware deve essere visto in modo uniforme dal software
- mantenere una interfaccia hardware-software uniforme

5.3 Funzionamento

Un speculatively multithread processor è un insieme di processing units o cores che lavorano assieme. Un sequencer ad alto livello spezzetta un programma in task e li assegna ai vari core. Un esempio di task è una porzione di istruzioni eseguite in sequenza, come il corpo di un ciclo o di una funzione. Ogni core esegue il task che gli è stato assegnato come farebbe un qualsiasi processore. Tutti i core che stanno eseguendo i vari pezzi di una finestra devono comunque mantenere un certo ordine di esecuzione delle istruzioni in modo da far sembrare sequenziale l'esecuzione delle istruzioni della finestra di cui fanno parte.

La caratteristica di essere una architettura distribuita permette la scalabilità del sistema nel momento in cui si vogliono aumentare il numero di core. Un altra caratteristica che permette di aumentare la scalabilità è il fatto che

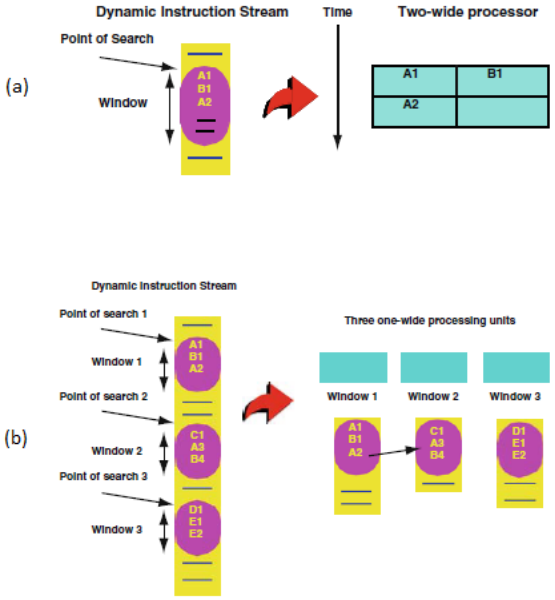

Figura 5.1: Suddivisione in task

si cerca il più possibile di evitare la comunicazione tra i vari core, e nel caso fosse necessaria, si cerca che questa avvenga il più possibile tra core vicini.

5.4 Modello di esecuzione

Le mutithread speculative architectures impiegano il compilatore o l'hardware per partizionare un programma sequenziale. Sebbene l'esecuzione delle attività non sia indipendente l'una con le altre, l'hardware ipotizza che lo siano e le esegue in parallelo. I task vengono quindi eseguiti in parallelo presupponendo che non abbiano dati da scambiarsi gli uni con gli altri. Appena il compilatore scopre la presenza di dipendenze tra i vari task, allora forzerà un meccanismo per correggere l'ordine di esecuzione. Se nonostante il controllo avvengono comunque delle violazioni di dipendenza, l'hardware rileverà queste violazioni e riavvolgerà l'esecuzione del task che ha violato la dipendenza e tutti i task che sono stati eseguiti dopo, in quanto potrebbero aver elaborato dei dati non validi. Ma come fa l'hardware/software a gestire tutto questo? Innanzitutto il sistema suddividerà il programma in task che possono essere eseguiti in parallelo, poi siccome alcuni task potrebbero avere delle dipendenze, servirà un modo per lanciare task secondo il giusto ordine di esecuzione ed infine, ogni core dovrà tenere in un buffer lo stato della memoria e dei registri per aspettare il giusto momento di fare il commit di

queste informazioni. Un requisito chiave del sistema è il fatto che bisogna tracciare l'ordine dei vari task. Infatti quando l'esecuzione di un task è terminata, è di fondamentale importanza sapere qual'è il task che deve essere eseguito immediatamente dopo. Questo ordine tra i task è mantenuto mettendo i core in un anello e assegnando i task man mano che devono essere eseguiti al prossimo core che si incontra sull'anello. [2]

Capitolo 6

General purpose multicore processors

6.1 Introduzione

Nel 1965, Gordon Moore predisse che in ogni nuova generazione della tecnologia sarebbero stati messi nello stesso pezzo di silicio due volte il numero di transistor che non nella generazione precedente. Negli anni passati questa legge è sempre stata verificata. Purtroppo questa crescita vertiginosa delle performance ha spinto molte architetture al loro limite. In particolare, lo spaventoso aumento della frequenza di clock, ha portato alla costruzione di chip incredibilmente complessi e che consumavano una notevole quantità di energia. A livello fisico, questo aumento indesiderato del consumo di energia ha condotto ad aumenti di costo nei dispositivi di erogazione dell'energia e di dissipazione del calore sviluppato. Infine si è anche raggiunto il limite di quanta potenza dare una CPU per essere venduta e comunque guadagnarci qualcosa. Nonostante tutto la legge di Moore non è ancora stata infranta, infatti continua a fornirci sempre più transistor nello stesso spazio. I progettisti quindi hanno posto l'attenzione anche al miglioramento delle prestazioni della memoria. Nel corso della storia infatti il divario tra periodo di clock e accesso alla memoria è aumentato vertiginosamente, creando lunghe latenze ogni volta che il processore deve accedervi. Per questo si è pensato di usare le memorie cache in modo gerarchico, migliorando così le prestazioni.

6.2 Sistema

Il consumo di energia è sempre stato un grosso problema da risolvere, anche perchè spesso i sistemi in cui bisogna installare un processore multicore hanno

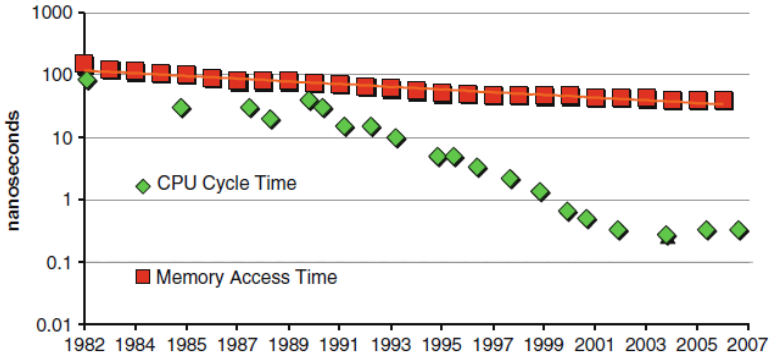

Figura 6.1: Divario

anche dei limiti prefissati. Inoltre quando si inseriscono più core nello stesso chip, bisogna ricordare che l'energia va divisa tra tutti, ed essendo questi i componenti che consumano la maggior parte di energia, è chiaro che bisogna porre molta attenzione anche a questo aspetto. Così si potrebbe diminuire il voltaggio al quale lavora la circuiteria, diminuendo drasticamente il consumo di potenza, ma in questo modo si diminuirebbe anche la frequenza di clock. Per questo si è pensato a mettere a lavorare assieme più core ma con frequenze minori in modo da avere buone prestazioni e bassi consumi.
Il kernel del sistema operativo è responsabile della gestione delle risorse e della maggior parte delle interfacce verso l'hardware presente nel sistema. Un sistema operativo può essere eseguito in ogni processore presente nel sistema, o anche in più di uno contemporaneamente, per questo bisogna introdurre dei metodi per proteggere il sistema da accessi incontrollati alle risorse, i quali porterebbero a dei danneggiamenti del sistema. Per questo sono stati introdotti dei sistemi di blocco degli accessi alle risorse, ma mano a mano che i core aumentavano, questa soluzione cominciava a mostrare problemi di scalabilità. Grazie all'aumento delle prestazioni generali, è diventato possibile suddividere le risorse presenti così da usarle in modo più efficiente in base alle situazioni di utilizzo. La virtualizzazione è una tecnica grazie alla quale è possibile suddividere le risorse fisiche del sistema a vari sistemi virtuali che sono isolati tra loro. Grazie a questa tecnica è ad esempio possibile eseguire più sistemi operativi nello stesso istante.

6.3 Caches

Poiché il numero di core presenti sullo stesso chip stanno aumentando di molto, bisogna porre l'attenzione anche alla gestione delle memorie cache. La

quantità di memoria di questo tipo che è possibile mettere a disposizione su un chip varia in funzione del numero di core presenti, dello spazio disponibile, e anche di eventuali budget di costo. Un'altra considerazione da fare è come distribuire al meglio la cache nella gerarchia, in base al fatto che è possibile che possa essere condivisa da più core. Tradizionalmente nei sistemi con un solo core si cerca di avere una cache di alto livello il più grande possibile, così da aumentare la probabilità che i dati che servono al processore siano già al suo interno. Questa soluzione è stata adottata anche nei processori multicore, infatti ogniuno di essi ha una cache L1 e L2 privata mentre una L3 condivisa tra tutti perchè se ad esempio più thread devono accedere alla stessa struttura dati, in questo modo l'accesso è più veloce. Questa condivisione può portare anche a funzionamenti poco performanti nel caso servano porzioni di memoria diverse per ogni core, in questo caso infatti la condivisone porta a rallentamenti. [2] [8]

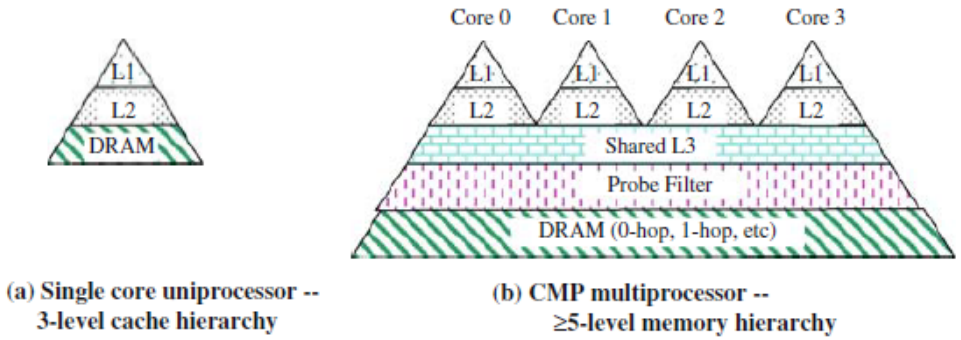

Figura 6.2: Gerarchia cache

Capitolo 7

BlueGene/L

7.1 Introduzione

BlueGene/L di IBM è un progetto risalente al 1999 che aveva lo scopo di risolvere problemi che richiedono una massiccia mole di calcoli, come ad esempio quelli delle scienze biologiche. Un problema delle scienze biologiche è quello di rappresentare la struttura tridimensionale di una proteina, perchè è proprio negli errati ripiegamenti di alcune di queste che determinano alcune malattie come la fibrosi cistica. Oltre a problemi di questo tipo, BlueGene/L può essere usato per problemi come la dinamica molecolare, modelli climatici, astronomia e modelli finanziari. Nel 2001 IBM cominciò a costruire questo nuovo supercomputer, ma con una nuova filosofia, infatti per i vari core, non vennero usati dei processori creati appositamente ma vennero usati i PowerPC 440, dei chip commerciali, non molto potenti ma con un basso consumo di potenza. Il primo chip venne alla luce nel giugno 2003. Il primo quarto del sistema BlueGene/L, costituito da 16.384 nodi, divenne pienamente operativo a partire dal novembre 2004 e, grazie ai suoi 71 teraflop/s, fu insignito del premio di supercomputer più veloce al mondo. Il consumo di soli 0,4 MWatt gli garantì anche il premio di supercomputer più efficiente della sua categoria con 177,5 megaflop/watt. La consegna della parte restante del sistema, che lo ha portato ad un numero di 65.536 nodi di calcolo, fù ultimata durante l'estate del 2005.

7.2 Nodi

Ogni nodo è costituito da due core PowerPC 440 a 700MHz. Ogni core ha due unità a virgola mobile che possono emettere quattro istruzioni a virgola mobile per ogni ciclo. I due core presenti in ogni nodo sono esattamente

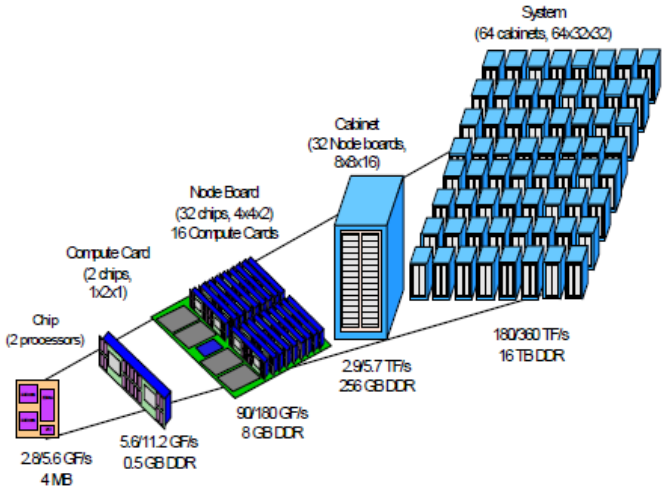

Figura 7.1: Struttura BG/L

uguali, ma sono usati per due differenti scopi, uno viene usato per il calcolo, mentre l'altro è usato per la comunicazione con gli altri 65536 nodi. Nel chip sono presenti tre livelli di cache, le cache L1 da 32KB per i dati e 32 per le istruzioni, le cache L2 da 2KB ed infine le cache L3 da 4MB.

7.3 Topologia

La connessione di tutti questi chip richiede una rete d'interconnessione scalabile e molto performante. La topologia usata è quella di un toro. La rete a toro è stata pensata per un utilizzo generale per comunicazioni punto-punto o multicast. La comunicazione all'interno del toro avviene mediante l'instradamento cut-through. Inoltre sono presenti anche altre quattro reti, una rete ad albero per riunire tutti i nodi, alcune applicazioni di BlueGene/L richiedono la partecipazione di tutti i nodi (ad esempio trovare il valore minimo tra 65536 valori diversi), e questa struttura semplifica il lavoro. La terza rete è usata per le barriere globali e la gestione degli interrupt, questa serve agli algoritmi che lavorano in più passi e magari hanno bisogno che termini una fase prima di cominciarne un'altra. La quarta e la quinta rete sono entrambe Gigabit Ethernet e servono a connettere i nodi di I/O ai server file (esterni a BlueGene/L) e quindi ad internet. L'altra rete Gigabit Ethernet serve per operazioni di debug. [9] [10]

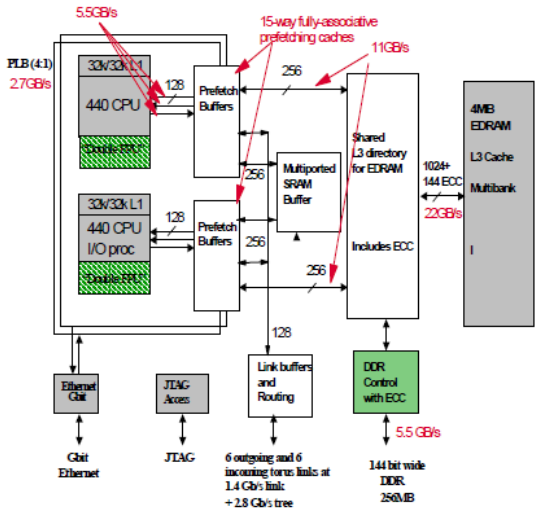

Figura 7.2: Struttura nodo

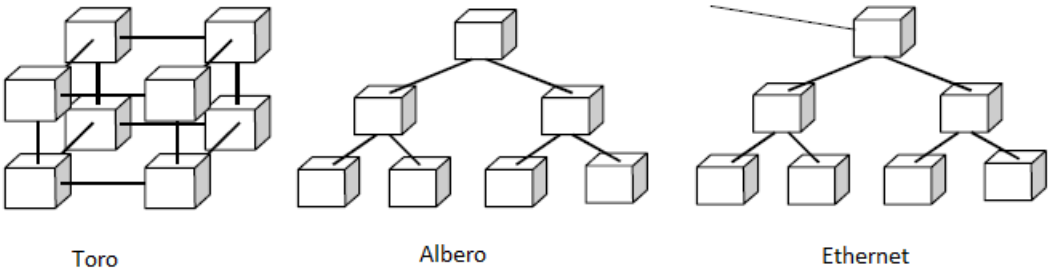

Figura 7.3: Mettere descrizione

Capitolo 8

Opteron

8.1 Introduzione

L'AMD Opteron è una serie di processori prodotti a partire dal 2003 che hanno portato diverse innovazioni nel campo. Innanzitutto è stato pensato per poter eseguire anche programmi a 32 bit senza rallentamenti, diventando così compatibile anche con programmi già presenti sul mercato. Un'altra caratteristica è che può accerere a più di 4GB di RAM, che era uno dei limiti delle precedenti architetture. Nel corso degli anni successivi AMD ha continuato su questa strada, fino ad arrivare al 2007, anno nel quale AMD ha immesso nel mercato la terza generazione di AMD Opteron, un chip equipaggiato con quattro core.

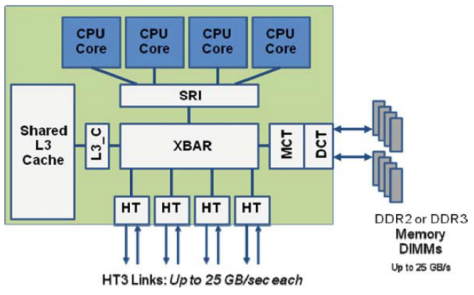

Figura 8.1: Opteron 4 core

8.2 Panoramica del sistema

Dalla seconda generazione di questo processore è disponibile la virtualizzazione di architetture di tipo x86. In questo modo può eseguire anche sistemi

operativi più datati o applicazioni che non supportano il suo nuovo tipo di architettura. Per quanto riguarda il consumo di energia, è stato introdotto un metodo che controlla il voltaggio di lavoro di ogni singolo processore, e quindi l'energia consumata, e in caso di necessità si riesce a regolare in modo automatico l'energia consumata facendo variare la frequenza di clock. La memoria cache è stata progettata in modo che ogni singolo core abbia una cache L1 da 64KB e una cache L2 da 512KB private, e una L3 condivisa da 6MB.

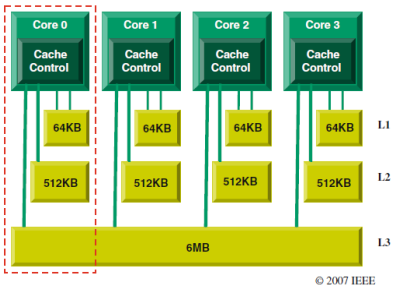

Figura 8.2: Gerarchia cache Opteron

Questo nuovo design ha anche introdotto un secondo DRAM controller, che riduce notevolmente i conflitti che possono crearsi tra i vari core, incrementando in questo modo l'efficienza complessiva. Un'altra miglioria introdotta è l'interfaccia HyperTransoprt, che permette di connetere i quattro core e inoltre può essere usato per gestirne anche un numero maggiore. Questo modo di connettere i vari core permette di avere un diametro della rete il più piccolo possibile, diminuendo così la latenza e anche il numero di nodi da attraversare per raggiungere la destinazione, inoltre i collegamenti tra i core sono usati più uniformemente, così da avere una migliore gestione del traffico. I miglioramenti apportati da questa topologia sono maggiormente visibili se nella rete sono presenti più core, in questo caso infatti la latenza diminuisce drasticamente. [2] [11]

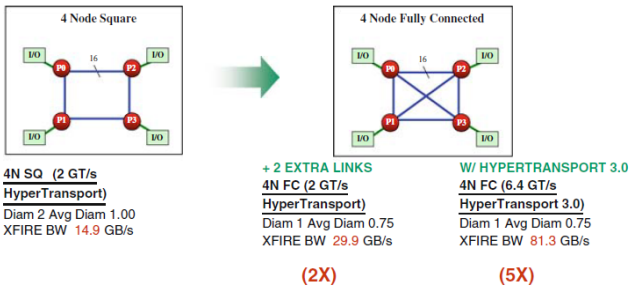

Figura 8.3: Topologia Opteron

Capitolo 9

Conclusione

Negli ultimi anni si sono raggiunti i limiti di potenza di calcolo che è possibile racchiudere in un singolo processore senza aumentare troppo la potenza assorbita o il costo di produzione e progettazione dello stesso. Per questo sono stati creati dei chip che fanno lavorare al loro interno più di un processore. In questo modo infatti è possibile avere le stesse prestazioni di un singolo processore molto potente ma con lo stesso consumo di energia. Purtroppo però questa nuova architettura ha sollevato anche diversi problemi che però sono stati risolti, come ad esempio quale sia il miglior modo per far comunicare i vari core, o come gestire l'esecuzione di un programma o anche quante e come assegnare le risorse disponibili all'esecuzione di un processo. Infatti se non gestiti al meglio, questi ed altri problemi possono causare un peggioramento drastico delle prestazioni, vanificando così gli sforzi fatti per aumentare le prestazioni facendo cooperare più core. In ogni caso questi problemi sono stati affrontati e risolti in modo magnifico, tanto che i processori multicore non sono usati solo in mercati di nicchia nei quali in caso di problemi c'erano degli esperti a risolverli, ma viene commercializzata anche al grande pubblico il quale è molto esigente riguardo a costo/prestazioni e affidabilità. Un'altra prova del successo di questa nuova architettura è che viene usata anche per costruire i supercomputer usati dalle più grandi e prestigiose associazioni del pianeta per risolvere problemi che richiedono una enorme potenza di calcolo. Questa nuova architettura sembra quindi una valida speranza per aumentare sempre di più la potenza di calcolo negli anni futuri e quindi la legge di Moore potrà sicuramente rimanere valida ancora per molto tempo.

Bibliografia

[1] D. A. Patterson and J. L. Hennessy, *Struttura e progetto dei calcolatori*. Zanichelli, 2010.

[2] S. W. Keckler, K. Olukotun, and H. P. Hofstee, *Multicore Processors and Systems*. Springer, 2009.

[3] A. B. Abdallah, *Multicore Systems On-Chip: Pratical Software/Hardware Design*. Atlantis Press, 2009.

[4] "Tilepro64tm processor product brief."

[5] S. Borkar and A. A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, no. 5, pp. 66–67, 2011.

[6] L. Hardesty, "Multicore may not be so scary," *MIT News Office*, 2010.

[7] L. Hardesty, "Chips as mini internets," *MIT News Office*, 2012.

[8] L. Hardesty, "The next operating system," *MIT News Office*, 2011.

[9] N. Adiga, G. Almasi, *et al.*, *An Overview of the BlueGene/L Supercomputer*. IEEE Computer Society Press Los Alamitos, 2002.

[10] A. S. Tanenbaum, *Architettura dei Calcolatori un approccio strutturale*. Pearson Prentice Hall, 5 ed., 2006.

[11] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway, *The AMD Opteron Processor for Multiprocessor Server*. IEEE Computer Society, 2003.