Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali

Corso di Laurea in Informatica

ANALISI E GESTIONE DEL PROTOCOL DEADLOCK NELLE NETWORK-ON-CHIP

Tesi di Laurea in Simulazione

Presentata da:
Simone Bernini

Relatore:
Luciano Bononi

Correlatore:
Nicola Concer

Sessione I
Anno Accademico 2009/2010

Parole chiave:
Network-on-Chip, NoC, deadlock, protocol-deadlock, message,dependent,deadlock

Prefazione

Il primo circuito integrato fu costruito nel 1958 da Jack St. Clair Kilby ed era composto da circa 10 componenti. Nel tempo questa tecnologia é stata sfruttata ed evoluta fino ad arrivare agli odierni 10.000.000 di transistor integrati in un unico chip.
I vantaggi di questo tipo di circuiti sono noti: la minore area utilizzata, la maggiore velocitá, i ridotti consumi e l'economicitá.
Nei chip appartenenti alla generazione ULSI (ultra-large-scale-integration) vengono integrati diversi sistemi complessi fino ad ottenere una architettura miniaturizzata con le stesse funzionalitá dell'architettura estesa. I sistemi integrati vengono normalmente connessi con un approccio bus-based, usando cioé un unico bus condiviso da tutti gli elementi come nell'architettura iniziale.
Qui subentra la novitá che ha mandato in fermento molte aziende del settore come Intel e STMicroelectronics ed altrettante universitá note come Stanford, Berkley, M.I.T e la stessa universitá di Bologna.
La grande novitá delle NoC sta nel sistema di interconnessione dei componenti, passando infatti da un approccio point-to-point (simile ad un grafo incompleto) si é giunti ad un sistema simile ad un circuito a commutazione di pacchetto derivato dal parallel-computing e basato su uno stack simile allo standard ISO-OSI.
Nei capitoli iniziali verranno introdotte alcune nozioni come quelle di architettura degli elaboratori e reti di calcolatori, necessarie per capire gli scenari antecedenti e successivi alle NoC, altre riguardanti le problematiche legate al deadlock, per comprendere uno dei problemi attorno al quale ruota la progettazione di questa tipologia di chip.
Nei capitoli successivi verranno presentate un po piú nel dettaglio le NoC con un approfondimento sul lavoro da me nello specifico approfondito: l'analisi del deadlock nelle Network on Chip e la sua gestione.

Buona Lettura.

Indice

Capitolo 1

Introduzione

In informatica, una condizione di deadlock e' una condizione di arresto, nella quale due o piu unitá non possono proseguire, perche in attesa di risorse.
Questo problema e' strettamente connesso all'esistenza stessa della multiplazione delle risorse e delle unita' di calcolo.
Per questo motivo il deadlock e' un problema presente ovunque vi siano multiple unita' di calcolo e risorse: dai supercomputer, alle reti globali il deadlock deve essere in qualche modo risolto. Raggiunti i limiti tecnologici prestazionali del singolo chip, l'evoluzione si sta spostando verso il calcolo parallelo, e questo lo si puo notare dalla recente introduzione di processori multi-core, che sembra la strada futura da percorrere, per continuare a seguire la legge di Moore.
Questo nuovo ramo tecnologico delle Network-on-Chip, si presenta come eccezionalmente innovativo, ed e' uno straordinario connubio tra diversi ambiti del mondo dell'informatica (microarchitetture,reti,sistemi operativi), nonostante presenti una serie di problemi vecchi e nuovi che devono essere affrontati sotto a una nuova luce. Tra questi problemi, il deadlock ricopre un ruolo fondamentale, perche' queste architetture, sotto certi versi, non fanno altro che esaltarne le caratteristiche problematiche, riportando questo argomento, forse leggermente abbandonato in questi ultimi anni, nuovamente allo studio.
Questa tesi si occupa di analizzare il problema, sotto questa nuova veste delle Network-on-Chip,esporre lo sviluppo attuale delle architetture, dal punto di vista della gestione del deadlock, e analizzare quelli che potrebbero essere i possibili sviluppi futuri per approcciarsi a questo problema.

1.1 Cenni di Architettura

1.1.1 Evoluzione della scala di integrazione

La produzione di circuiti integrati é stata resa possibile, nel corso del ventesimo secolo, grazie ai progressi nella ricerca sui semiconduttori. Questi hanno sostituito l'utilizzo delle valvole nella costruzione dei circuiti, permettendo alti gradi di affidabilita, prestazioni e abbassando il costo totale degli apparati.
La possibilita di produrre un circuito capace di contenere milioni di componenti é stato un enorme progresso rispetto all'assemblaggio manuale degli stessi. Inoltre la produzione di massa di componenti basati su semiconduttori, ha aumentato l'affidabilita, la standardizzazione, l'economicitá e la disponibilitá degli stessi. Tutt'oggi é piu conveniente comprare un piccolo chip che integra multiple funzioni e migliaia di transistor, piuttosto che un singolo transistor acquistato come componente singolo.
Le performance di semiconduttori come i transistor sono molto superiori a quelle di componenti come le valvole: il consumo é infinitamente inferiore, e la velocitá di switching[1] é estremamente piu elevata. Inoltre i transistor non hanno svantaggi quali il tempo di accensione, e la fragilitá intrinseca delle valvole.
I primi circuiti integrati erano di tipologia SSI (Small-scale-integration),prodotti negli anni 50: poche decine di transistor che implementavano una, o comunque pochissime funzionalitá. Questi circuiti sono stati indispensabili per la realizzazione del programma aerospaziale americano degli anni 50, e viceversa. La grande richiesta, da parte della NASA[2], di circuiti per i sistemi di navigazione, ha fatto si che lo sviluppo procedesse velocemente verso la fase successiva.
Il passo successivo dello sviluppo sono stati i chip MSI(medium-scale-integration): circuiti dalle dimensioni ridotte, rispetto agli SSI, e che implementavano maggiori funzionalita, a dispetto di un costo solo leggermente superiore.
A metá degli anni 70 si arrivó agli LSI (large-scale-integration), raggiungendo la quantitá di alcune migliaia di transistor per circuito. Questi permisero la produzione del primo processore e del primo banco di memoria RAM[3].
La produzione é continuata introducendo i chip VLSI (very-large-scale-integration), che continuano a essere prodotti ancora oggi. I chip hanno cominciato ad avere centinaia di migliaia di transistor, a partire da metá degli anni 80, fino ai diversi miliardi di transistor, presenti oggi in integrati attualmente in commercio. Sono stati vari, i fattori che hanno portato a questa crescita di integrazione, a partire dalla grande richiesta di microcontrollori. Nuovi processi produttivi quali

[1] la capacita di passare da uno stato all'altro.
[2] National aerospace agency
[3] Random access memory: memoria ad accesso casuale

la produzione di circuiti all'interno di ambienti controllati, e la sostituzione di tecnologie che introducevano consumi inadeguati di energia, hanno fatto si che milioni di transistor potessero essere condensati all'interno di pochi centimetri quadrati. Alcuni affermano che da quando si é arrivati ad integrati con numero di transistor superiore al milione, si dovrebbe passare all'acronimo ULSI (ultra-large-scale-integration), ma la notazione classica rimane comunque VLSI.
Un passo fondamentale in questo é stata l'adozione della fotolitografia, come metodo produttivo: non si produce un transistor come componente singolo, ma si costruisce una mappa dell'integrato, sulla quale successivamente un apposita macchina rimuovera, grazie all'ausilio del laser, strati successivi, formando lentamente l'intero chip come se fosse un singolo componente.

System-on-a-chip

Questo termine si riferisce alla produzione di un singolo chip che contiene tutte le componenti di un normale computer.Solitamente questi chip contengono componenti digitali, analogici, misti, e tipicamente sono in grado di operare in radiofrequenza, per quel che riguarda le comunicazioni.Un tipico utilizzo di questi sistemi é nel campo dei sistemi embedded[4].
Le differenze con i microcontrollori standard sono minime. Solitamente i SoC sono integrati di maggiore potenza e risorse, capaci di eseguire un sistema operativo al loro interno, mentre i microcontrollori end-user,solitamente usati nei personal computer, hanno bisogno di strutture aggiuntive quali RAM e dischi, mentre i SoC sono progettati per essere sistemi stand-alone. La differenza al giorno d'oggi é comunque minima, e il termine SoC, indica piú un processo produttivo direzionato all'integrazione dei componenti, alla miniaturizzazione e alla semplificazione, piuttosto che alla ricerca di prestazioni.

1.1.2 Bus di sistema

Un bus di sistema é una componente che consente, all'interno di un elaboratore, la comunicazione tra i componenti. Non si tratta di una connessione punto a punto, si tratta di un componente capace di interconnettere contemporaneamente molti dispositivi su un unico canale. Tipicamente il bus puó essere rappresentato come un unico filo di connessione tra componenti diversi.
Il bus di sistema presente in tutti gli elaboratori é composto da un certo numero di fili e di connessioni a intervalli regolari che permettono l'inserimento in appositi slot di nuove periferiche di input/output o memoria.

[4]sistemi computerizzati integrati, in grado di eseguire una o piu funzioni

Ognuno di questi fili é progettato per trasmettere informazioni logiche, e quindi cifre binarie, anche se non sono tutte della stessa categoria, infatti il bus di sistema é diviso in 3 sotto-bus:

- dati: bidirezionale, contiene i dati che si vogliono trasmettere da una periferica generica (anche la cpu) a un altra;
- indirizzi : monodirezionale, contiene l'indirizzamento fornito dalla cpu indispensabile per un input/output;
- controlli : monodirezionale, usato dalla cpu per effettuare controlli nella gestione delle comunicazioni sul bus stesso.

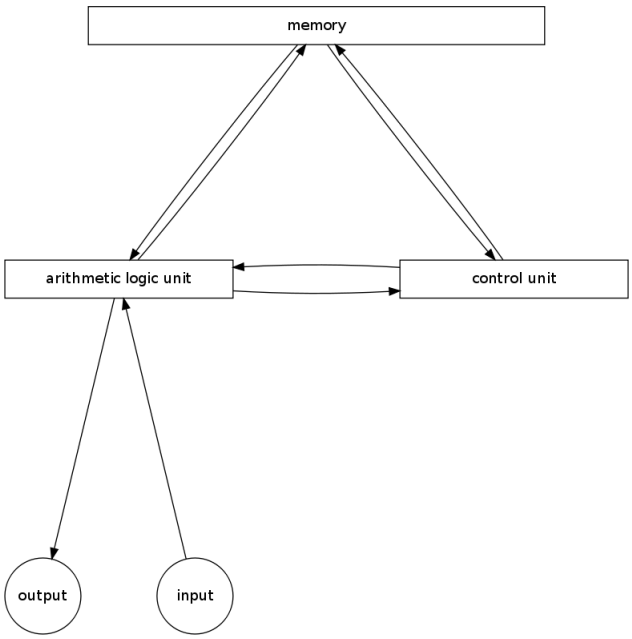


Figura 1.1: Macchina di Von Neumann

Nel modello originale dell'architettura a bus, progettato da Von Neumann, si prevedeva una gestione distribuita del bus, ed eventualmente la presenza di piu bus. Questo si é rivelato difficile da realizzare, mentre appare piu sensato avere una periferica, detta arbitro del bus che si occupa di gestire l'utilizzo di questo canale condiviso.

Nella gestione con arbitraggio del bus, l'arbitro del bus puó fornire a una periferica che lo richiede, l'utilizzo del bus, fornendo un segnale di GRANT che avvierá la comunicazione.
I protocolli fondamentali di gestione sono 2:

- Daisy chaining: se a una periferica arriva un segnale di grant, utilizza il bus, se ne ha bisogno. Se non necessita del bus, propaga il segnale di grant alla periferica successiva. Quindi per il principio di localita hanno maggiore prioritá le periferiche piu vicine fisicamente all'arbitro del bus;

- Indipendent request: ogni periferica ha una linea di grant che la connette all'arbitro indipendente. Se una periferica ha bisogno del bus per molto tempo, l'arbitro controlla che nessun altro necessiti del bus stesso, e in quel caso permette alla periferica di comunicare per tutto il tempo di cui ha bisogno.

Esistono anche tecniche di arbitraggio decentralizzato del bus, ma sono studi piu che altro accademici, e difficilmente vengono poi concretizzati in prodotti finiti. Questa tecnica peró é da notare come renda il sistema immune da guasti all'arbitro, rispetto a un sistema centralizzato, dove un guasto renderebbe l'intero calcolatore inutilizzabile.
Nelle architetture moderne, sempre alla ricerca delle prestazioni migliori, ci si é scontrati con il limite fisico di un unico bus di essere efficiente al crescere del numero di periferiche, e il problema dovuto all'avere periferiche come CPU e RAM, che necessitano spesso del bus di sistema per le proprie comunicazioni.
Per ovviare al problema, si sono creati canali dedicati che hanno permesso di migliorare le prestazioni: in un elaboratore moderno abbiamo 3 bus di sistema distinti:

- cpu-ram: bus dedicato ad alta velocita per queste due componenti che sono quelle piu utilizzate, in un sistema, e necessitano sia di grande bandwidth, che responsivitá;

- northbridge bus: connessione cpu-periferiche a medio-alta priorita, quali i dischi(ide,sata) o le periferiche connesse su slot pci(standard,pci-express,scsi);

- southbridge bus:connessione tra il northbridge bus e tutte quelle periferiche a basse prestazioni, o che non necessitano di risposta real-time, quali schede audio,lettori cd-rom,bus usb, e molti altri.

Questa tecnica ha aumentato di molto la complessita di un moderno elaboratore, pur aumentando le prestazioni. Questo significa anche un aumento dei costi di

produzioni, e controlli molto complicati per quello che riguarda l'arbitraggio di questi bus multipli.

1.1.3 limiti dell'architettura classica nell'ULSI

Chiaramente questo approccio non potrá essere adottato all'infinito: all'aumentare delle periferiche, aumentare il numero di bus diventa sempre piu sconveniente, e le prestazioni tendono a crollare, in quanto un gran numero di periferiche sullo stesso bus soffre di starvation, e il sistema precipita in quello che é un trashing[5] hardware.
Per prima cosa le periferiche a maggior prioritá rubano troppo tempo a quelle con minore prioritá, impedendogli di prendere possesso del bus per tempo abbastanza lungo.
Come secondo aspetto, poi, l'aumentare della complessitá dell'arbitro del bus, poi, introduce ritardi dovuti al maggiore tempo dovuto a decidere quale periferica prenderá controllo del bus. In pratica a lungo andare, si passa piú tempo a decidere piuttosto che all'effettivo tempo di computazione utile alle trasmissioni, e questo fenomeno é molto simile al thrashing nella gestione della memoria principale, in un sistema operativo.

[5] fenomeno per il quale si passa piu tempo dovuto alla gestione della paginazione, piuttosto che all'effettiva computazione dei processi, in un sistema operativo

1.2 Cenni di Reti

1.2.1 Reti a commutazione di pacchetto

Una rete di computer é un insieme di elaboratori interconnessi tra loro e capaci di comunicare secondo determinati protocolli. Gli elaboratori possono essere connessi tramite diversi mezzi di trasporto, come cavi o onde radio, e possono essere distribuiti su una superficie piu o meno vasta: da pochi metri a migliaia di chilometri.
Le odierne reti di elaboratori sono dette a commutazione di pacchetto, tale tecnica consente la gestione di un canale condiviso (un cavo, o l'etere) tra piu stazioni in modo non deterministico.I pacchetti sono le unita di base che viaggiano sulla rete e in cui sono suddivise le informazioni da trasmettere. Un informazione qualsiasi viene divisa in pacchetti, che vengono inviati sulla rete in maniera asincrona e su percorsi a volte differenti, e poi vengono riassemblati per rigenerare l'informazione iniziale, una volta giunti a destinazione.
Per la loro natura, le reti a commutazione di pacchetto (che comprendono tutte le reti formate da elaboratori che non siano apparati telefonici), sono soggette a due inconvenienti:

- perdita di pacchetti
- ritardi

Per quanto riguarda la perdita di pacchetti, si verifica quando un pacchetto viene ricevuto danneggiato, e quindi viene scartato, oppure quando gli apparati di commutazione si trovano con i buffer in ingresso e in uscita pieni, e sono costretti a scartare i pacchetti in arrivo.
I ritardi possono essere causati da vari fattori, ma fondamentalmente il ritardo totale di un pacchetto é la somma di quattro componenti fondamentali:

- elaborazione
- propagazione
- trasmissione
- coda

Il ritardo di elaborazione comprende il tempo di analisi di un pacchetto da parte di qualsiasi commutatore (l'elaboratore stesso, o gli apparati lungo la linea) per decidere l'instadamento, e comprende il tempo di controllo degli errori sul pacchetto stesso.

Il ritardo di propagazione riguarda il mezzo fisico, e rappresenta il tempo fisico in cui un mezzo (come ad esempio il rame), riesce a trasmettere un impulso elettrico.
Il ritardo di trasmissione é legato alla banda disponibile, ed é minore quando la velocitá intrinseca della linea, é maggiore.
Il ritardo di coda é il ritardo dovuto al mezzo trasmissivo occupato attualmente da altri pacchetti, o dovuto al fatto che altri pacchetti hanno una maggior precedenza rispetto a quello attuale.Dei 4 tipi di ritardo, questo é l'unico non prevedibile, e che quindi é soggetto ad analisi statistiche.

1.2.2 Lo stack ISO-OSI

Il modello iso/osi é uno standard nato oltre 30 anni fa, che definisce una serie di strati per fornire un modello astratto per quel che riguarda il networking e la comunicazione. Gli strati definiti sono 7:

- **fisico** gestisce la trasmissione di flussi di bit non strutturati su una linea di comunicazione;
- **data link** raccoglie un flusso di bit in un contenitore piú grande denominato trama;
- **network** gestisce l'instradamento dei pacchetti in una rete;
- **trasporto** gestisce il canale logico fra mittente e destinatario;
- **sessione** usato per aggregare diversi flussi di trasporto;
- **presentazione** si occupa del formato dei dati scambiati tra piú controparti;
- **applicazione** consumatore dei dati.

Il punto focale del modello a strati é fornire una serie di strati successivi, ognuno dei quali risolve problemi legati alla comunicazione, e presenta al livello superiore un astrazione libera da quei problemi.Inoltre ogni strato, in trasmissione, incapsula e manipola le informazioni in modo di trasferire sul supporto fisico un pacchetto ben formato e che rispetta il protocollo di comunicazione. All'arrivo, il pacchetto passerá tutti gli strati del modello, in senso contrario, subendo in senso opposto tutte le modifiche e ritornando l'informazione originaria immutata.
Per la trattazione che ci interessa, gli strati piú importanti sono i primi 4,gli ultimi tre strati si occupano di fornire al sistema operativo e alle applicazioni,

primitive ad alto livello per le comunicazioni, tendenzialmente affidabili e che non si occupano del mezzo di trasporto o della qualita della connessione,o della gestione degli errori.

Hub

E' un apparato che lavora al livello 1 dell'iso/osi, e che consente la connessione di molti computer, tramite cavi ethernet (IEEE 802.3),facendoli agire come se comunicasseto tutti contemporaneamente sullo stesso canale.
Quello che fa questo apparato é ripetere ogni singolo segnale su ogni porta, permettendo a tutti gli elaboratori coinvolti di comunicare.Gli hub sono semplici ripetitori, non si occupano di gestire il traffico o altro, semplicemente ripetono su ogni porta ció che gli arriva dai terminali.Come side-effect di questo, hanno il pregio di aiutare a gestire le collisioni, ripetendo un segnale di jamming su tutte le porte, e facendo quindi capire a tutti i pc che é avvenuta una collisione.

Switch

E' un apparato che lavora al livello 2 dell'iso/osi, e che consente la connessione di molti computer, tramite cavi ethernet (IEEE 802.3), creando di volta in volta connessioni punto-a-punto tra due computer che vogliono comunicare.Lo switch gioca una parte attiva, nelle comunicazioni, tiene delle tabelle per conoscere gli indirizzi MAC presenti su ogni porta, tiene dei buffer per migliorare la qualitá della comunicazione, e adotta tecniche di store/and/forward per il controllo dell'errore. In generale uno switch, migliora di molto le prestazioni in una rete, rispetto ad un hub, sebbene la loro funzione sia la stessa. Inoltre la banda totale di un hub é rappresentata dal valore massimo al quale un terminale puó comunicare con un altro(o il doppio, se full duplex), perché una stazione, in trasmissione, occupa tutte le altre porte. In uno switch invece ogni stazione che comunica puó raggiungere il massimo della capacitá in banda dello switch (full-duplex).

Router

E' un apparato che lavora al livello 3 dell'iso/osi, e che consente la connessione tra reti diverse di computer, gestendo l'instradamento dei pacchetti. Mantiene in memoria una tabella per gestire reti diverse, che in questo modo possono comunicare tra loro.Il router apre i pacchetti in ingresso, controlla la rete di destinazione, e invia il pacchetto nella giusta porta.

1.2.3 Topologie di rete

Una topologia di rete é la disposizione fisica dei computer facenti parte della rete stessa, e il modo in cui sono interconnessi.
Solitamente vengono rappresentate tramite grafi connessi, in cui ogni nodo rappresenta un elaboratore, o comunque un apparato di rete.
Le varie topologie si possono dividere in cinque grandi gruppi:

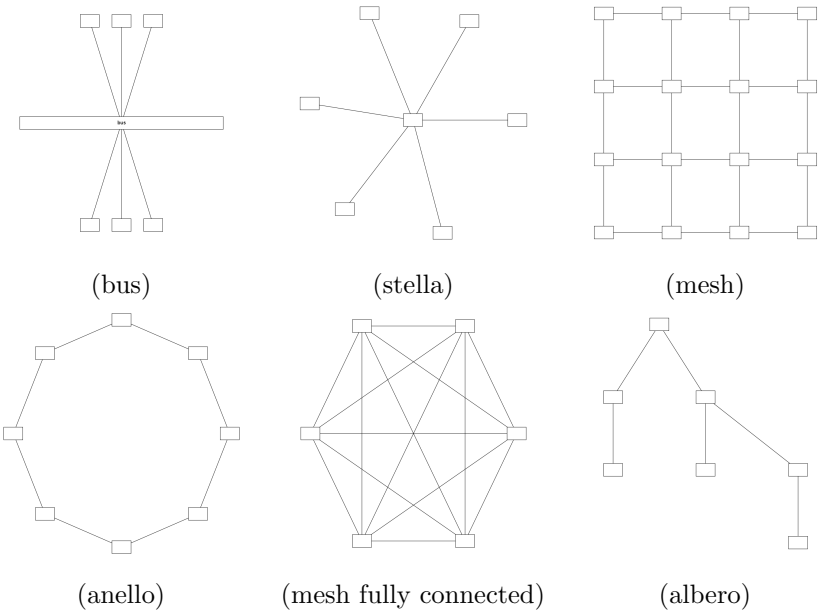


Tabella 1.1: Topologie Standard Di Rete

- bus;
- stella;
- mesh;
- anello;
- albero.

Bus: ogni macchina é attaccata con un cavo ad un canale comune, detto bus, ai cui capi sono presenti terminatori, per non fare rimbalzare il segnare elettrico dagli estremi. Questa architettura, sebbene poco costosa, é piuttosto inefficente infatti, le collisioni sulla linea sono parecchie, solo un elaboratore alla volta puo fare uso del canale e la scalabilita della rete é pressoche assente. Un guasto sul bus principale crea gravi problemi in questa rete.

Stella: ogni macchina é connessa a un centro, solitamente rappresentato da un hub o uno switch. Il vantaggio di questa rete, pur rimanendo centralizzata, é che ogni nodo puó creare una connessione punto a punto con qualsiasi altro nodo della rete, ottimizzando le prestazioni. Uno switch di centro stella, inoltre migliora le prestazioni, permettendo la comunicazione contemporanea di piu nodi. Aggiungere altri elaboratori, in questa rete, é piuttosto semplice, ma lo svantaggio riguarda possibili rotture nel centro-stella, che invalidano completamente l'utilizzo della rete.

Mesh: le reti mesh sono un tipo di topologia nella quale ogni nodo puó agire come router indipendente. Questo tipo di topologia consente un riconfigurarsi continuo della rete, per aggirare nodi non piú raggiungibili o congestionati. Questa topologia differisce dalle altre perché permette a ogni nodo di essere connesso con un qualsiasi altro numero di nodi(da 1 solo a n, dove n é il numero totale di nodi della rete). Questo tipo di rete, sebbene presenti problemi di gestione e di routing piuttosto complessi, é di gran lunga piu affidabile di qualsiasi altra topologia di rete: ogni volta che c'é un problema, con opportuni metodi puo essere aggirato. Tipicamente queste reti somigliano molto come problemi ad alcuni tipi di reti wireless ad-hoc, che introducono inoltre la mobilitá dei nodi nello spazio, come problema aggiuntivo. Una topologia mesh puó essere completamente connessa, quando ogni suo nodo é connesso a ogni altro nodo, o parzialmente connessa, dove i vari nodi hanno un numero variabile di connessioni.

Anello: ogni macchina é connessa alla precedente e alla successiva, formando un anello.Una comunicazione verso un nodo puó avvenire in un verso o nell'altro. Questa topologia di rete é spesso legata allo standard token ring, per la quale é stato sviluppato, in modo da ottimizzare la gestione delle collisioni. Il problema piú grave di questa rete é in caso di disconnessione/crash di un nodo, momento in cui l'intera rete smette parzialmente/totalmente di funzionare.

Albero: é un tipo di rete gerarchica, in cui i nodi non sono a pari livello di importanza. Il centro della rete é un nodo radice, al quale via,via attacchiamo nodi figlio, ai quali sono connessi altri nodi e cosi via.Una rete ad albero deve avere almeno 3 livelli di ramificazione, altrimenti é funzionalmente equivalente a una rete di tipo stella. Chiaramente il nodo radice é il fulcro della rete, e una sua compromissione si ripercuote su tutte gli elaboratori. D'altro canto é piú tollerante ai guasti rispetto a una normale rete a stella, pur mantenendo bassi i costi e permettendo maggiore efficienza.

Ogni nodo puó ramificarsi in N altri nodi figli, e questo N rappresenta il fattore di ramificazione. Piu N é basso (fino a uno), piú la topologia tende ad essere equivalente a una struttura a bus.

Virtual Channel

E' un tipo di approccio chiamato anche modello orientato alla connessione. Prima di inviare i dati verso un host, deve essere effettuata una connessione virtuale con l'host stesso. Quindi la connessione é effettuata in due parti: la prima parte é l'instaurazione della connessione, e la seconda riguarda il trasferimento effettivo dei dati.
Per effettuare un trasferimento, bisogna inserire in un apposita tabella(detta tabella dei VC) lo stato di connessione; questo é composto da una serie di informazioni quali:

- id di circuito virtuale, che identifica in uno switch univocamente una connessione
- un interfaccia di ingresso attraverso cui i pacchetti arrivano allo switch
- un interfaccia di uscita attraverso cui i pacchetti escono dallo switch
- un altro identificatore di circuito virtuale, eventualmente diverso, che servira per i pacchetti uscenti

Una connessione virtuale viene identificata univocamente dalla combinazione del VCI dei pacchetti che vengono ricevuti dallo switch e dall'interfaccia da cui vengono ricevuti. Naturalmente é possibile per ogni switch avere molte connessioni virtuali instaurate allo stesso tempo. In generale poi, i valori di VCI in ingresso e uscita tendono ad essere diversi, per cui un valore di VCI di per se non ha il significato di identificare univocamente una connessione, ma ha significato solo data una certa linea di connessione.
Grazie a questa tecnica ogni switch presente fra il sorgente e la destinazione puó effettuare una sorta di multiplexing sui canali fisici di interconnessione, vedendone quindi un numero maggiore. Viene quindi ad essere virtualizzato il numero di canali presenti fra uno switch e l'altro.

1.3 Cenni di Concorrenza

In informatica possiamo vedere la concorrenza come una situazione che si verifica quando piú processi o piú unita di elaborazione, sono in esecuzione nello stesso istante e necessitano, per far evolvere la loro elaborazione, dell'accesso a risorse condivise.
Come sappiamo, all'interno di un elaboratore, convivono contemporaneamente molte attivitá, e alcune di queste, se non gestite correttamente, porterebbero a un inconsistenza dei dati, con tutto ció che ne potrebbe conseguire.
Per introdurre la concorrenza é necessario introdurre il concetto di processo e il concetto di thread.

- Processo: In informatica un processo é un istanza di un programma, che consiste in uno o piú thread,sequenzialmente eseguiti da un elaboratore in grado di eseguire piú programmi contemporaneamente.
- Thread: Un thread é una singola linea esecutiva all'interno di un programma.

Da questo si puó desumere che un processo é formato da threads. In pratica un thread é l'unita granulare con cui possiamo suddividere un processo.Ogni thread puó essere eseguito in maniera parallela agli altri, e un processo ha sempre almeno un thread (se stesso), inoltre vi sono casi in cui un processo puó avere piuú thread eseguiti in parallelo.
In inglese thread é qualcosa di simile a filo, rende bene l'idea immaginare una corda come un processo, e i singoli fili sono i thread che la compongono e che si susseguono in parallelo.
Un processo si puó descrivere completamente con alcuni parametri:

- il suo contatore di processo (program counter);
- il suo stato del processore (registri);
- il suo stato in memoria (dati,codice,stack);
- le strutture utilizzate (file in uso);

Inoltre un processo si puó trovare, in un dato momento, in uno dei tre stati:

- *running*, cioé attualmente in esecuzione, il processo possiede il processore e tutte le risorse fisiche e di input/output di cui ha bisogno;
- *ready*, cioé il processo possiede tutte le risorse fisiche e di input/output di cui ha bisogno;

- *waiting*, cioé il processo é in attesa di almeno un input/output;

1.3.1 Parallelismo

La concorrenza é quindi una situazione contrapposta al concetto di sequenzialita, e introduce a sua volta i concetti di:

- parallelismo virtuale
- parallelismo reale

Nel primo caso, il parallelismo virtuale, abbiamo una singola unitá di elaborazione,nella quale convivono piú processi per dare l'idea di un esecuzione contemporanea di piú processi sullo stesso elaboratore. In questo caso si parla di interleaving, e i processi sono alternati nel tempo.L'introduzione dell'interleaving solitamente introduce latenza a causa della gestione aggiuntiva dei processi.
Nel secondo caso,il parallelismo reale, abbiamo multiple unitá di elaborazione che consentono l'esecuzione effettiva di piú elaborazioni allo stesso tempo.In questo caso si parla di overlapping, e i processi sono alternati nello spazio.

Un sistema informatico si dice multiprogrammato quando premette parallelismo virtuale.
Un sistema informatico si dice multiprocessore quando premette parallelismo reale.
Tutti i moderni sistemi informatici permettono la convivenza di entrambi i parallelismi, sullo stesso elaboratore.
Quindi la concorrenza si occupa di descrivere e risolvere i problemi legati all'esecuzione contemporanea di piú processi.Questi sono legati solitamente alla sincronizzazione dei processi stessi, o alla comunicazione tra processi.

1.3.2 Race Condition

Un semplice esempio che dimostra problemi legati all'esecuzione parallela di piú processi:

```
int a=10
int b=5

process A(){
a+=b;
b+=5;
```

```
}
```

Se avviamo 2 o piú processi A(), il valore finale dei due parametri a,b, varierá a seconda dell'avvicendamento dei singoli processi tra loro.
Questa circostanza viene denominata race condition, ed é una spiacevole situazione che si verifica quando il risultato di un insieme di operazioni dipende dalla temporizzazione con la quale queste operazioni vengono eseguite.
Le race condition vanno evitate.

1.3.3 Deadlock

Un altro fatto da tenere in considerazione é l'interazione tra processi: i processi possono cooperare per raggiungere in modo piú efficiente a un risultato comune, oppure i processi possono comunicare per scambiarsi informazioni.
I sistemi operativi devono fornire primitivi per permettere questo tipo di operazioni, nella fattispecie devono fornire primitive di sincronizzazione e di comunicazione.
Le risorse sono unitá alle quali i processi accedono per leggere/scrivere dati di cui necessitano. L'accesso a una risorsa si dice mutualmente esclusivo se in ogni istante, solo un processo puó accedere a quella risorsa.
In certi momenti é necessario garantire,per la correttezza del risultato:

- la *mutua esclusione* nell'accesso a una determinata risorsa;
- l'*esecuzione atomica* (senza interruzioni possibili) di un set di istruzioni;

Soddisfatte queste due condizioni, necessarie per la correttezza del risultato di un set di operazioni, emergono due gravi problemi per i quali la programmazione concorrente deve trovare soluzione:

- deadlock
- starvation

Quando due treni convergono a un incrocio, ambedue devono arrestarsi, e nessuno dei due puó ripartire prima che l'altro si sia mosso.

Legge Del Kansas di inizio XX secolo.

Un deadlock é una condizione definitiva di blocco di un insieme di processi, in cui ognuno aspetta risorse dagli altri.

Altre situazioni particolari oltre al deadlock

Starvation:

Per starvation si intende l'impossibilita per un processo di accedere nel tempo a una determinata risorsa (o a un set di risorse), non a causa di un blocco del sistema (deadlock), ma semplicemente perche gli altri processi, a causa dell'esecuzione temporale, riescono ad occupare quelle stesse risorse piú velocemente.
Nell'esempio dei filosofi a cena, é possibile che un filosofo non mangi mai, perche tutti gli altri riescono a mangiare sempre prima di lui.
Solitamente la causa della starvation risiede nell'algoritmo di scheduling, che non assegna in modo equo le risorse di sistema ai processi, e permette che alcuni di questi siano privilegiati rispetto ad altri. In uno scheduling a priorita, ad ogni processo é associato un livello di priorita che gli permette di procedere nella sua esecuzione piu velocemente rispetto agli altri processi. Questo aumenta la reattivitá del sistema e permette a processi importanti di procedere piú in fretta. L'altra faccia della medaglia, in un sistema del genere, é la comparsa di possibile starvation nel sistema: i processi a bassa priorita non verranno mai serviti, se quelli ad alta prioritá continuano a venire schedulati.
Per risolvere i problemi legati alla starvation, solitamente basta implementare una politica di scheduling fair, cioé che tratta paritativamente tutte le entitá che richiedono risorse, come il round robin (quanti di tempo predefiniti e equivalenti assegnati a ogni processo).
Ad oggi si adottano sistemi misti (round robin di code di prioritá, con prioritá dinamiche per processo), che possono permettere sia grande responsivitá in un sistema (processi importanti che procedono velocemente) sia un controllo piu o meno buono della starvation.

Nei sistemi wireless (e non) é possibile avere starvation: in particolari algoritmi di selezione dei nodi che cercano di massimizzare il throughput (Maximum throughput scheduling algorythm), i nodi con il minor CNR (carrier to noise ratio), cioe i nodi con il minor disturbo sul segnale, vengono visti come nodi che possono parlare gratis, cioe hanno una prioritá maggiore rispetto a nodi con alte interferenze.

Livelock

Il livelock é uno speciale caso di deadlock, in qualche modo legato alla starvation. Si tratta infatti di starvation legata alle risorse che non riescono a essere prese da processi.Si tratta di una condizione nella quale due o piú processi continuano a cambiare di stato uno rispetto all'altro, senza realmente procedere

nell'esecuzione, ognuno a causa dell'altro(degli altri) processo/i.
In generale ś una condizione che puó essere rappresentata come due persone in un corridoio: ognuna si sposta da un lato per far spazio all'altra, ma nessuna delle due passa, per lasciar passare prima l'altra. Il corridoio(la risorsa) non viene mai occupato, e le persone(i processi), non procedono nell'esecuzione.

1.3.4 Deadlock Avoidance e Recovery

L'esistenza del deadlock é legata a 4 condizioni fondamentali:

- mutua esclusione;
- hold and wait;
- assenza di preemption;
- attesa circolare;

Venendo a mancare una di queste condizioni, viene a mancare la presenza di deadlock in un sistema.

Vediamole nel dettaglio:

Mutua esclusione: le risorse sono ad accesso seriale,cioé in ogni istante di tempo, al piú un processo puo accedere a quella risorsa. Per garantire la coerenza nei risultati la mutua esclusione deve essere presente, ed alcune risorse (eg stampanti) sono obbligatoriamente da serializzare. Nel caso di risorse che non devono essere serializzate, e si vuole comunque attaccare questa condizione del deadlock, esistono algoritmi, detti algoritmi di sincronizzazione non bloccanti, che impediscono il bloccarsi di piu processi in attesa di risorse, a causa della mutua esclusione;

Hold and Wait: le risorse possono essere richieste da un processo anche durante la sua esecuzione. Il caso ideale sarebbe far richiedere ad ogni processo tutte le risorse di cui avra bisogno per la sua esecuzione, prima che cominci l'elaborazione.Purtroppo é una strada non sempre praticabile, perche non si puó conoscere a priori le risorse, e il numero di queste che un processo occupera.Inoltre questo sistema presenta una grande inefficienza di utilizzo delle risorse;

Assenza di Preemption: le risorse non possono essere tolte a un processo durante la sua esecuzione. Per eliminare il deadlock é una condizione

difficile da trattare: un processo potrebbe aver bisogno di utilizzare la risorsa per certi periodi di tempo, per giungere a risultati utili, oppure il suo stato potrebbe diventare inconsistente, o il sistema potrebbe finire in thrashing*;

Attesa Circolare: due o piú processi possono formare una catena circolare nella quale ogni processo attende risorse dal successivo. Un sistema per impedire questa condizione é assegnare precedenza a ciascuna risorsa, e si forzare i processi a chiedere risorse in ordine di priorita (un processo che possiede una risorsa con precedenza k non puó chiedere altre risorse con precedenza minore di k).Un altro approccio (inefficienete)é permettere l'allocazione di una sola risorsa alla volta: un processo libera la risorsa attuale per accedere alla successiva.

Rilevare un deadlock é, in generale, un problema piu semplice da risolvere, piuttosto che prevenirlo. Prevenire un deadlock prima che questo accada é un problema indecidibile (lo stesso problema della fermata puó essere riformulato, sullo schema di un deadlock). Controllare che ci sia un deadlock,una volta che si é verificato, é piu semplice per via del fatto che il sistema operativo, ad ogni momento, conosce lo stato di assegnazione di tutte le risorse di sistema, divise per processo, e le mantiene in apposite strutture.
Una volta rilevato un deadlock sono disponibili varie strade per procedere:

- *terminare tutti i processi coinvolti* garantisce la ripresa del sistema, ma é molto oneroso da pagare per processi che stanno elaborando da molto tempo e sono importanti;
- *terminare alcuni dei processi coinvolti fino a che il deadlock non sparisce* come sopra, garantisce a un certo punto la ripresa del sistema, ma é comunque molto oneroso;
- *effettuare un rollback del sistema* riportando il sistema a uno stato sicuro é fattibile, ma comunque si perde parte del lavoro fatto dai processi, e non ci garantisce che il problema non si ripresentera.Inoltre é necessario tenere una storia esecutiva completa per ogni processo, aumentando di molto l'overhead di sistema.

Il deadlock puó essere evitato se si conoscono preventivamente alcune informazioni sui processi, ed é per questo che non é sempre possibile adottare questa tecnica. L'idea é che ogni volta che un processo richiede risorse, si controlla che la richiesta, insieme a tutte le altre delle risorse allocate, non comporti della caduta del sistema in uno stato unsafe, dove unsafe significa la possibilita che,

con un certo insieme di richieste successive, si verifichi un deadlock.
Un tipico esempio di algoritmo é il cosiddetto algoritmo del banchiere. Tale algoritmo pone come presupposto il conoscere in anticipo il massimo numero di richieste di risorse che un processo puó effettuare. In alcuni sistemi é un informazione impossibile da ottenere, ed é per questo che l'algoritmo non é sempre applicabile. L'algoritmo mette in pausa e ritarda richieste che potrebbero portare il sistema in uno stato unsafe, e cerca di trovare una possibile serie di allocazione di risorse tale da soddisfare tutti i processi.
Il comportamento dell'algoritmo é la rappresentazione a chiave informatica di una banca: i clienti sono i processi, le risorse sono la liquidita bancaria, ogni cliente ha un tetto massimo di richiesta di prestito. Chiaramente la banca tiene conto dei prestiti effettuati e del suo credito residuo, e deve elargire prestiti in maniera controllata, perche se ogni cliente chiedesse il suo massimale di prestito, la banca non sarebbe in grado di soddisfare tutti. L'algoritmo cerca di tenere il sistema in uno stato safe, dove safe significa che esiste una sequenza di allocazioni di risorse che ci permette di soddisfare tutti con certezza. É da notare che uno stato unsafe non é garanzia di deadlock, ma solo possibilitá.
Esistono molti algoritmi in grado di evitare il deadlock, molti di questi sono basati sulla frenata simmetrica di processi. Alcuni di questi, come wait/die , wound/wait, tengono conto della gerarchia di vita dei processi (padre/figlio), per cercare di evitare una situazione di deadlock e decidere a chi permettere di allcoare risorse. Un approccio molto usato nei sistemi moderni é l'algoritmo dello struzzo: cercare di controllare/evitare/prevenire un deadlock puó essere molto costoso dal punto di vista delle risorse e delle prestazioni del sistema, quindi si ignora semplicemente il problema dando per scontato che non si verificherá mai, anche se comunque si adottano alcune prevenzioni nei kernel dei sistemi operativi per evitare condizioni difficili.
Introdurremo brevemente alcune strutture per gestire la mutua esculsione. L'obiettivo di queste strutture é garantire il rispetto della mutua esclusione all'interno del codice concorrente che fa uso di sezioni critiche e potrebbe essere soggetto a race conditions.Verranno solamente citate per dare un esempio di come un problema di concorrenza puo' essere risolto.Introdurremo:

- *Test and Set*
- *Semafori*
- *Monitor*
- *Message Passing*

Test and Set

Istruzioni semplici, implementabili in hardware. Riescono in maniera atomica (senza interruzioni), ad effettuare due operazioni (p.e. test and set), utili per controllare l'ingresso in una sezione critica.
Si utilizzano due tipi di variabile: una globale e visibile da tutti i processi, e una locale a processo. Questo tipo di struttura e' comunque soggetto a busy-waiting,ma garantisce il rispetto di una sezione critica.

```
boolean global = false

   P(){
1   while TS(global)
2     wait
3   CS
4   global = false
   }
```

Semafori

Un semaforo é una variabile protetta che, come le Test and Set, permette di risolvere il problema delle sezioni critiche.
E' composto principalmente da due istruzioni, P e V che incrementano/decrementano un contatore interno, che indica l'utilizzo di una risorsa.Presentano comunque il problema del busy waiting.

```
Semaphore s;

   P(){
1   S.p();
2   CS
3   S.v();
   }
```

Quando il contatore scende a zero, la chiamata .p diventa bloccante, consentendo di implementare la mutua esclusione.

Monitor

I monitor sono astrazioni ad alto livello creati per risolvere i problemi di semafori e TS.
L'idea é avere oggetti che risolvano il problema della mutua esclusione e al

contempo siano capaci di mettere in attesa e far ripartire i processi, a richiesta. Un monitor é composto da dati privati, un suo costruttore/inizializzatore, una serie di procedure visibili all'esterno.
Un processo agisce sul monitor solo invocando le procedure, che provvederanno ad aggiornare i contatori interni, inoltre solo un processo puó accedere in un dato momento al monitor, tutti gli altri processi sono sospesi. Da questo punto di vista é concettualmente molto simile agli oggetti visti in programmazione a oggetti.
Un monitor e' composto da due primitive fondamentali:

- wait
- signal

Un processo che fa wait su una condizione, attende,sospendendosi, il verificarsi di quella condizione,mettendosi in un apposita coda.
Un processo che fa una signal su una condizione, segnala che quella condizione é verificata, al processo in cima alla coda di attesa.Il processo che ha fatto la signal viene messo in pausa aspettando che il nuovo processo riattivato termini la sua esecuzione (politica signal-urgent).
Oltre alla politica signal-urgent, esistono molte altre politiche di gestione.

```
class Buffer {
  buf = new double;
  notFull = new ConditionVariable();
  notEmpty = new ConditionVariable();
}

public synchronized void deposit(double data) {
  if (count == 0) wait(notFull);
  buf = data;
  signal(notEmpty);
}

public synchronized double fetch() {
  double result;
  if (count == 1) wait(notEmpty);
  result = buf;
  signal(notFull);
  return result;
}
```

1.3.5 IPC e Message Passing

In tutti gli altri costrutti, la comunicazione tra processi avviene tramite memoria condivisa. Nel caso del message passing, il costrutto implementa un tipo di comunicazione tra processi. Chiaramente questa comunicazione é mediata dal sistema operativo, per implementare la protezione della memoria.
I processi in gioco, si scambiano messaggi formattati per informarsi del proprio stato, e per scambiarsi dati. Quello che fa il costrutto é spostare dei dati da uno spazio di memoria di un processo allo spazio di memoria di un altro processo.
Le due primitive fondamentali di questo paradigma sono:

- **send** serve per inviare messaggi;
- **receive** serve per riceverne;

Come sono implementate queste operazioni determina il funzionamento del paradigma. Premettendo che un processo, per ricevere, deve obbligatoriamente eseguire una receive, possiamo implementare send e receive come bloccanti, quindi un processo che effettua una send, attende che il processo a cui ha inviato faccia una receive,e un processo che fa una receive rimane bloccato in attesa di ricevere qualcosa. Questo message passing é detto totalmente sincrono.
Possiamo utilizzare una send non bloccante, dove chi fa receive rimane in attesa, mentre chi fa send prosegue nell esecuzione, implementando un message passing parzialmente asincrono.
Possiamo utilizzare send e receive non bloccanti, implementando un message passing totalmente asincrono.

	send	receive
sincrono	bloccante	bloccante
asincrono	non bloccante	bloccante
totalmente asincrono	non bloccante	non bloccante

Tabella 1.2: Caratteristiche dei vari tipi di message passing

Nelle receive é possibile specificare un mittente, oppure usare una wild-card per specificare un mittente qualsiasi e ricevere cosi, da tutti.
Il problema piu evidente dell'utilizzare message passing é dato dalla perdita di prestazioni che si puó avere in un sistema. Infatti quelli che ci si scambia, sono spazi di indirizzamento, e non dati, generando in questo modo un aumento di

overhead[6]aggiuntivo che puó peggiorare l'efficienza di un sistema. Il message passing, comunque, é un costrutto per implementare la mutua esclusione molto elegante, e, nonostante generi una perdita prestazionale, é tra i costrutti piu utilizzati, se non quello piu diffuso.
Due costrutti presentano lo stesso potere espressivo quando ognuno dei due puó esprimere ogni tipo di programma espresso nel linguaggio dell'altro.
Nel caso delle strutture che implementano la mutua esclusione, ci troviamo di fronte a costrutti quali TS(ed equivalenti),semafori e monitor, che presentano esattamente lo stesso potere espressivo, per quanto riguarda i metodi a memoria condivisa.
E' da notare che il message passing di tipo asincrono é piu espressivo, di quello sincrono.Infatti implementare il secondo con il primo é molto semplice, mentre il contrario diventa impossibile, a meno di non utilizzare strutture di appoggio, quali strutture dati aggiutive e processi.

[6]Banda richiesta oltre ai dati necessaria per gestire la comunicazione.

Capitolo 2

Network on Chip

Le Network on chip[2][10][11] (in seguito NoC) sono un esempio di Ultra Large Scale Integration Chip, difatti in un singolo chip vengono installati diversi sistemi complessi con tutta la logica del sistema iniziale e l'architettura per la comunicazione fra questi.
Nelle architetture moderne[38], sempre alla ricerca delle prestazioni migliori, ci si é scontrati con il limite fisico di un unico bus di essere efficiente al crescere del numero di periferiche, e il problema dovuto all'avere periferiche come CPU e RAM, che necessitano spesso del bus di sistema per le proprie comunicazioni. Per ovviare al problema, si sono creati canali dedicati che hanno permesso di migliorare le prestazioni: in un elaboratore moderno abbiamo 3 bus di sistema distinti:

- il cpu-ram bus, dedicato ad alta velocita per queste due componenti che sono quelle piu utilizzate, in un sistema, e necessitano sia di grande bandwidth, che responsivitá;

- il northbridge bus connette cpu-periferiche a medio-alta priorita, quali i dischi(ide o sata) o le periferiche connesse su slot pci(standard,pci-express,scsi);

- il southbridge bus, connessione tra il northbridge bus e tutte quelle periferiche a basse prestazioni, o che non necessitano di risposta real-time, quali schede audio, lettori cd-rom, bus usb, e molti altri.

Questa tecnica ha aumentato di molto la complessita di un moderno elaboratore, pur aumentando le prestazioni. Questo significa anche un aumento dei costi di produzioni, e controlli molto complicati per quello che riguarda l'arbitraggio di questi bus multipli.

Chiaramente questo approccio non potrá essere adottato all'infinito: all'aumentare delle periferiche, aumentare il numero di bus diventa sempre piu sconveniente, e le prestazioni tendono a crollare, in quanto un gran numero di periferiche sullo stesso bus soffre di starvation, e il sistema precipita in quello che é un trashing[1] hardware. Per prima cosa le periferiche a maggior prioritá rubano troppo tempo a quelle con minore prioritá, impedendogli di prendere possesso del bus per tempo abbastanza lungo. Come secondo aspetto, poi, l'aumentare della complessitá dell'arbitro del bus, introduce ritardi dovuti al maggiore tempo dovuto a decidere quale periferica prenderá controllo del bus. In pratica a lungo andare, si passa piú tempo a decidere piuttosto che all'effettivo tempo di computazione utile alle trasmissioni, e questo fenomeno é molto simile al thrashing nella gestione della memoria principale, in un sistema operativo.

La vera novitá sta nel sistema di interconnessione fra i chip in quanto si é passati da un sistema bus-based ad un poco scalabile sistema point-to-point fino ad arrivare alle NoC: reti a commutazione di pacchetto e basate su uno stack simile all'ISO/OSI. Riassumento, le NoC, non sono altro che dei System-on-Chip nei quali é stato completamente riprogettato il sottosistema di comunicazione, diventando simili a una rete vera e propria: una NoC é costruita su link point-to-point multipli, interconnessi da router i quali, esistendo multipli percorsi che interconnettono due nodi, si occupano di gestire l'instradamento dei pacchetti. Nonostante un apparente somiglianza tra le NoC e le normali reti, esistono delle differenze.
Le NoC, da un punto di vista teorico, possono essere viste esenti da perdita di pacchetti, dal disturbo dei segnali, cosi come con latenza definita e prefissata su tutti i percorsi. Questo sappiamo non essere vero nelle reti reali basate su ethernet, dove viene effettuato un continuo controllo di correttezza e di arrivo dei pacchetti.

2.1 Vincoli progettuali

Nelle NoC, per motivi progettuali, la semplicitá é un requisito fondamentale: l'apparato deve essere implementato direttamente in hardware e non si possono implementare funzionalitá troppo complesse direttamente nei circuiti tenendo bassi i costi e mantenendo un alto grado di flessibilitá.
Il basso consumo é una richiesta: minore calore dissipato dai componenti significa maggiore efficienza e tasso di integrazione piú alto facilmente raggiungibile

[1] fenomeno per il quale si passa piu tempo dovuto alla gestione della paginazione, piuttosto che all'effettiva computazione dei processi, in un sistema operativo.

(una maggiore integrazione a sua volta riduce la latenza).
La riusabilitá dei componenti é fondamentale: le NoC sono divise a strati, come abbiamo visto. Poter riutilizzare gli stessi micro-apparati in contesti che variano permette di tenere bassi i costi di produzione e quindi, in un secondo momento, aumentare l'efficienza e le prestazioni.
Chiaramente esistono dei trade-off. Circuiti e topologie semplici, difficilmente raggiungeranno l'efficienza o le funzionalitá di componenti piu avanzati, a fronte peró di un costo molto inferiore.
La topologia della NoC, inoltre, é un fattore molto importante: alcune topologie privilegiano la latenza, altre il throughput, mentre altre hanno addirittura nodi con algoritmi di routing diversi.
Quindi, come mostrato, nella progettazione di una NoC, devono essere tenuti in considerazione una miriade di aspetti:

- costo;
- semplicitá;
- riusabilitá;
- consumi;
- topologia (influenza la latenza, il throughput e altri aspetti prestazionali).

2.2 Stack e Paradigma di comunicazione

Come abbiamo detto lo stack usato nelle NoC é simile a quello ISO/OSI e lo presentiamo qui senza scendere nel dettaglio:

- Fisico: riferito a tutto quello che concerne ai dettagli elettrici, ai circuiti e alle tecniche per guidare i pacchetti;
- Data Link: assicura una trasmissione affidabile e gestisce la concorrenza per la condivione dei canali;
- Rete: gestisce le caratteristiche legate alla topologia(Mesh, FBFLY, Ring, Crossbar etc.) e fornisce l'algoritmo di routing;
- Trasporto: gestisce i servizi end-to-end e la segmentazione dei pacchetti;
- I livelli superiori possono essere accorpati nel livello Applicazione, una sorta di livello di interfaccia ai servizi, esposti all'infrastruttura tramite una paradigma Message Passing.

Il paradigma di interconnessione fra i componenti richiede la progettazione di nuovi protocolli. Nonostante le NoC possano esser viste come un sistema parallelo questi sistemi sono caratterizzati da stretti vincoli sugli algoritmi (visto che sono implementati in HW) e sulle risorse utilizzabili.
Le NoC richiedono quindi lo sviluppo di nuovi algoritmi e protocolli per risolvere:

- Routing: Gli algoritmi vengono implementati come circuiti integrati nella NoC. Devono esser semplici, veloci e devono sfruttare un numero di buffer ridotto al minimo. Per questa ragione adottano uno schema per inoltrare pacchetti detto wormhole. Wormhole consente l'uso delle pipeline e riduce il costo dei buffer. I pacchetti vengono divisi in unitá dette flits; le code in ogni nodo hanno la granularitá di un flit e i collegamenti fisici node-to-node sono gestiti da un controllo di flusso che lavora su una base flit-per-flit.

- Deadlock: viene risolto prendendo spunto dale soluzioni adottati nei sistemi distribuiti reali come i Virtual Channel ma questa é una soluzione costosa in quanto molti VC implicano molti buffer quindi una maggiore area occupata con una sensibile dissipazione di potenza e deve essere ridotta al minimo indispensabile.

- Mapping: I componenti vengono connessi in base al pattern di traffico che generano. In questo modo é possibile ottenere variazioni di performance anche nell'ordine del 250% e la dissipazione di potenza nell'ordine del 60%.

- Condivione dei dati: risolvendo il problema del deadlock al livello rete non si é comunque evitato il problema delle dipendenze circolari a livello trasporto. Questo tipo di deadlock viene definito message-dependent-deadlock o protocol deadlock e si scatena quando un Processing Element(PE) e uno Storage Element(SE) condividono lo stesso canale di comunicazione per inviarsi messaggi.

- Sincronizzazione del sistema: Le NoC possono essere implementate come sincrone(tutti i nodi lavorano con uno stesso segnale di clock), semi-sincrone(tutti i nodi lavorano con lo stesso segnale ma possono sfruttare fasi diverse) o completamente asincrone.

2.3 Topologie

Un punto chiave che influisce sulle prestazioni di una NoC, é la topologia di connessione delle componenti.
Una topologia deve essere semplice: questa caratteristica é indispensabile per

contenere i costi di produzione e la complessitá degli algoritmi di routing. Inoltre la complessitá di un circuito influisce pesantemente sulla massima frequenza utilizzabile, e quindi sulla velocitá massima ottenibile dal circuito stesso.
Chiaramente ogni tipo di topologia presenta dei punti di forza e dei punti di debolezza, e lo studio é tutt'ora aperto per trovare topologie con buone prestazioni in tutti gli aspetti che riguardano la NoC.

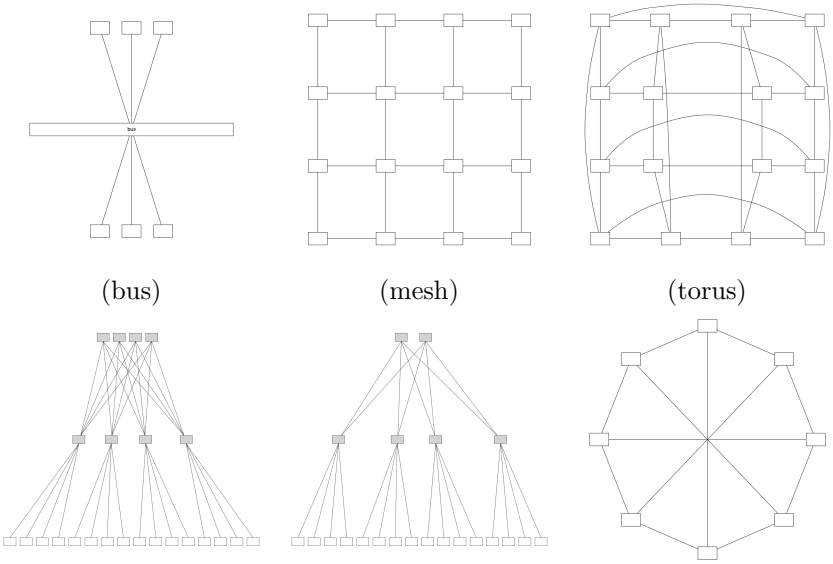


Tabella 2.1: Topologie Standard Di Rete

mesh : Questa architettura consiste in una n x m rete mesh di nodi interconnessi da router. Ogni nodo é connesso al proprio router, e ai router adiacenti. Quindi ogni nodo, eccetto quelli agli angoli, é connesso ad altri 4 nodi, ed il numero di router é uguale al numero di nodi. Ogni link tra due router consiste in un doppio link, rappresentante un doppio canale di comunicazione.

torus : Questa architettura consiste in una rete mesh, come presentata sopra, tranne per il fatto che i router ai lati sono connessi ai router sul lato opposto. Il numero totale di router rimane invariato (uguale al numero di nodi), e aumenta il numero delle connessioni tra nodi. La particolaritá é che, in questo modo, ogni nodo ha lo stesso numero di connessioni con altri nodi (4), e quindi il routing risulta piu semplice. Lo svantaggio consiste nel ritardo introdotto dalla connessione tra lati, che presenta un filo piú

lungo, rispetto agli altri, e quindi, maggior ritardo. Questo puó essere evitato dalla topologia folded torus che peró non viene quí presentata.

spin : In questa topologia di rete, ispirata alle topologie ad albero, i nodi sono a gruppi di quattro, e per ogni 4 nodi é presente un router. Ogni router connesso ai nodi, é duplicato a livello della radice. I 2 livelli di router (alla radice e ai nodi) sono completamente interconnessi. La rete in questo modo cresce, dati N nodi, con velocita (NlogN/8), e il numero di router converge a 3N/4.

butterfly fat tree : É una topologia di tipo ad albero, che usa i nodi come foglie, e mette i router come radici. I nodi sono raggruppati secondo un parametro: tipicamente sono raggruppati quattro a quattro, connessi allo stesso router. Ogni router é connesso poi a router di livello superiore, in numero minore al livello inferiore, in modo da rendere la rete completamente interconnessa. Il numero di livelli influenza il numero di connessioni e il numero di nodi radice a livello zero, che si occupano di interconnettere tutti gli altri router in gioco. Un nodo é identificato da una coppia (lvl,num) dove lvl rappresenta il livello di profonditá dell'albero, e num rappresenta il numero di nodo in quel livello. Quindi per ogni livello abbiamo una numerazione progressiva dei nodi li presenti. In questa rete il numero di router converge, per un numero arbitrario di nodi, a una costante, che é N/2, dato N numero di nodi.

spidergon : Una topologia di rete proposta da ST Microelectronics, rappresenta una variante di una topologia standard ad anello, in cui i nodi opposti dell'anello sono interconnessi. In questo modo ogni nodo possiede un router interconnesso sempre con altri 3 router: due adiacenti e uno opposto. I nodi presentano tutti uguale tipo di connessioni e per questo l'algoritmo di routing é equivalente per tutti. Inoltre la rete mantiene una certa semplicita, abbattendo i costi e migliorando le prestazioni.

La scelta della topologia alla quale puo essere legata un architettura e' una componente fondamentale della sua progettazione e caratterizza l'architettura dei componenti dalle quali e' composta in modo unico.Sebbene esistano topologie "generiche" come ad esempio la topologia mesh, ogni specifica architettura viene simulata solitamente sulla topologia per la quale e' stata pensata.Naturalmente esistono altri tipi di topologie oltre a quelle qui presentate, ma queste sono solitamente quelle piu utilizzate e che presentano un'ampia varieta' di situazioni possibili.

2.4 Componenti

Essendo un campo di ricerca in fase di sviluppo, non esiste ancora uno standard chiaro e definito per assemblare una NoC.In altre parole, ogni produttore definisce arbitrariamente un set di componenti che compongono la NoC, ed eventualmente e' possibile, tra NoC distinte, che un componente con lo stesso nome, non svolga la stessa funzione.
Vedremo piu avanti degli esempi e dei confronti tra varie architetture.
Introduciamo man mano i vari componenti, partendo dai due principali: router e network interface.

2.4.1 Router

I router sono un componente fondamentale delle NoC.Si occupano di gestire l'invio dei flit attraverso l'architettura.La loro progettazione e' strettamente connessa alla topologia della rete, e di fatto presentano un numero di porte di ingresso e uscita pari al numero di piste alle quali sono fisicamente collegati per comunicare. Lo scopo principale del router, universalmente condiviso nelle architetture di tipo NoC, e' implementare ed eseguire la funzione di routing per la quale e' stato progettato.Tale funzione garantisce l'arrivo dei pacchetti a destinazione attraverso un percorso calcolato,e possibilmente dovrebbe cercare di gestire il deadlock di tipo routing. I router,poi, utilizzando appositi buffer, ricevono in ingresso i vari pacchetti provenienti dalla NoC, e li gestiscono in base alla loro tipologia, in modo da ricevere tutti i flit di cui e' composto un pacchetto, oppure fare forwarding di pacchetti che non appartengono al nodo corrente.

2.4.2 Network Interface

La Network Interface(NI), nella sua accezione generica, si occupa della gestione del traffico end-to-end in uscita da un nodo.Solitamente presenta un interfaccia utilizzabile dagli strati superiori, e che rappresenta l'interfaccia per accedere al sottosistema di comunicazione della SoC, che e' rappresentato proprio dalla NoC.I pacchetti in arrivo dagli strati superiori, vengono solitamente frammentati in unita' piu piccole (dette flit) dalla NI, che si occupa di formattarli in un ben preciso formato e di passarli al componente successivo (il router) per l'invio attraverso la rete.
Allo stesso modo la NI colleziona i flit in arrivo dal router in modo da ricomporre i pacchetti inviati da altri nodi.
La NI, poi, puo' presentare un qualche meccanismo di flow control che permette

di regolare la quantita' di traffico immesso sulla rete in base alla saturazione della rete stessa.

Capitolo 3

Analisi e Gestione del Protocol Deadlock nelle Network on Chip

3.1 Introduzione ai deadlock sulle NoC

Nei sistemi paralleli, l'efficienza e l'affidabilita sono indispensabili per le prestazioni.Gli sviluppi tecnologici recenti hanno permesso un enorme incremento nella parallelizzazione, e l'emergere di nuove applicazioni affamate di banda richiede prestazioni sempre migliori. Per gestire efficacemente una rete parallela, bisogna gestire efficacemente il deadlock.Il deadlock si verifica come dipendenza circolare tra varie risorse, causata da messaggi in trasmissione. Per quanto sia importante gestire il deadlock comunque non bisogna imporre parametri troppo restrittivi nella rete, che potrebbero penalizzare l'utilizzo delle risorse.

3.1.1 Deadlock nelle Network on Chip

Per quanto riguarda il definire il deadlock, si fara' riferimento a una sintassi usata da Dally e Seitz in un paper sul routing deadlock[1], adattandola a un generico deadlock(adatto, quindi, a descrivere il Protocol Deadlock).
Prendiamo per assunto che:

- Un messaggio che arriva a una destinazione e' eventualmente consumato
- Un nodo puo' generare messaggi per un qualsiasi altro nodo sulla rete

- Il percorso che deve fare un messaggio e' determinato dalla sua destinazione, e non da altri fattori.(in altre parole si tratta di routing deterministico, non adattivo)

- Un nodo puo' generare messaggi di varia lunghezza, ma comunque solitamente maggiore di una singola unita' di trasmissione(flit).

- Una volta accettato il primo flit, il canale aperto deve accettare il resto del messaggio, prima di passare a un nuovo messaggio(wormhole routing)

- Una coda disponibile puo' arbitrare tra messaggi che l'hanno richiesta,ma non sceglie chi tra le code di messaggi in attesa deve trasmettere

- I nodi possono produrre messaggi a una velocita' indefinita,anche se limitata dallo spazio presente nelle varie code

In seguito diamo delle definizioni per descrivere la rete:

Definizione 1 Una rete di connessioni e' un grafo direttamente connesso, $I = G(N, C)$.
L'insieme degli angoli del grafo, N, rappresenta l'insieme dei nodi, mentre l'insieme degli spigoli, C,rappresenta l'insieme dei canali di comunicazione. Associata a ogni canale c_i, vi e' una coda di capacita' cap(c_i).
In una comunicazione per un canale c_i il nodo sorgente e' detto s_i e il nodo destinazione e' detto d_i

Definizione 2 Una generica funzione di movimento $F : CxN \implies C$ mappa il canale corrente c_c, e il nodo di destinazione, n_d, al successivo canale c_n, passando per il route definito dalla funzione,da c_c a n_d con $F(c_c, nd)$.
A un canale non e' permesso di passare da se stesso ($c_c \neq nd$).La cosa particolare e' che le successioni di movimenti che compie un messaggio fino alla destinazione sono senza memoria, cioe' non si ricordano del passaggio precedente, ma questa definizione permette di rendere i vari passaggi dipendenti dai canali, piuttosto che dai nodi, rispetto ad altri tipi di definizioni possibili (es. $NxN \implies C$).
E' da notare che una funzione di Routing e' un sottoinsieme di tutte le possibili funzioni di movimento contenute in F.Questa definizione rende quindi il concetto di movimento dei flit indipendente dal routing, e permette di esprimere anche il concetto di message dependent deadlock tramite questa definizione.

Definizione 3 Un grafo di dipendenza delle risorse(ad esempio, i canali di

comunicazione), D, data una rete di connessioni, I, e una funzione di movimento, F, e' un grafo diretto, $D = G(C, E)$.
Gli angoli di D sono le risorse della rete di connessioni I, mentre gli spigoli di D sono le coppie di risorse connesse tra loro dalla funzione che definisce i movimenti, F.

$E = \{(c_i, c_j)|F(c_i, n) = c_j \text{ for some } n \in N\}$

Definizione 4 Una configurazione e' un assegnamento di una lista di nodi per ciascuna coda.Il numero di flit presenti in ogni coda per il canale c_i e' dato dalla funzione size(c_i).
Se il primo flit della coda per il canale c_i ha come destinazione il nodo n_d, allora head(c_i)=n_d.
Una configurazione e' valida semplicemente quando:

$\forall c_i \in C, size(c_i) \leq cap(c_i)$

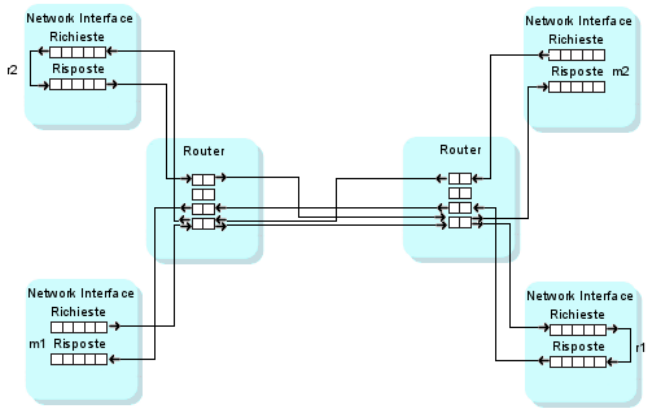
Cioe' vale che per ogni coda, il numero di flit contenuti, non sia superiore alla dimensione della coda stessa.

Definizione 5 Si definisce cosi' una configurazione di deadlock per una data funzione di movimento F, come una configurazione non vuota e valida dell'insieme delle varie code dei canali tale che:

$\forall c_i \in C, (head(c_i) \neq d_i and c_j = F(c_i, n) \implies size(c_j) = cap(c_j)$

Questo significa che in questa precisa configurazione, nessun flit e' a distanza di un passo dalla sua destinazione, e nessun flit puo' avanzare a causa del fatto che il buffer del canale successivo e' pieno.Una funzione di movimento F e' priva di deadlock su una data rete I, se non esiste alcuna configurazione che puo' presentare questo stato.

3.1.2 Message-dependent Deadlock



Abbiamo visto come il deadlock possa essere un evento catastrofico, e la minaccia di un deadlock puo' essere una seria minaccia ai servizi offerti dalla NoC.
Per questo motivo lo studio del deadlock e la ricerca di sistemi per gestirlo e' un punto chiave nella ricerca sulle NoC[2].
Molto lavoro e' stato focalizzato sullo sviluppo di NoC esenti da problemi quali il routing-deadlock.
Il Message dependent deadlock(o protocol deadlock) e' un problema attuale, ma lo sviluppo tecnologico sulle NoC e' ancora per molti aspetti in fase di ricerca, e vi sono addirittura delle Network on Chip che non risolvono il problema del MDD [3], oppure lo risolvono solo parzialmente tramite limitazioni sui protocolli che e' possibile usare[4].
Assumiamo che la nostra NoC sia composta solamente da una serie di router ai quali sono associate delle Network Interface.I router possono essere collegati in qualsiasi modo tra loro, e con le NI.
Assumiamo inoltre che la nostra rete sia libera da problemi di deadlock a livello routing. Le NI, come abbiamo visto, forniscono servizi end-to-end, semplificando la trasmissione point-to-point tra i vari nodi.
Per le nostre definizioni usiamo la sintassi definita in altri paper[5]. la quale distingue, in una comunicazione end-to-end tra NI, master e slave.I master sono NI che iniziano una transazione tramite richieste, gli slave ricevono e soddisfano la transazione.Eventualmente la transazione richiede una risposta, formulata dallo slave, sotto forma di un qualche tipo di dato, o un messaggio di acknowledgement.
Come si vedra' questo tipo di definizione portera' a quattro tipi distinti di dipendenze: richiesta-risposta, risposta-richiesta,richiesta-richiesta,risposta-risposta,

dipendentemente da come e' stato strutturato il protocollo della NoC.

La terminologia usata per rappresentare le dipendenze tra messaggi puo' essere presa da un paper[6] che analizza il problema e tratta di possibili soluzioni in ambito di supercomputer e sistemi paralleli.
Definiamo una catena di dipendenze come una lista parzialmente ordinata di tipi di messaggi.
Questi tipi di messaggi sono definiti da msg_1 a msg_n quindi la lunghezza della catena,implicitamente,definisce il numero massimo di tipi di messaggi, e definiamo $msg_i < msg_j$ se e solo se un messaggio di tipo msg_j puo' essere generato da un nodo produttore di messaggi di tipo msgi.
Un protocollo formato da n tipi di messaggi e' detto n-way protocol, e l'unico messaggio che ha una funzione diversa, e' il tipo di messaggio msg_n, detto terminante perche e' ricevuto, e non genera una risposta aggiuntiva.

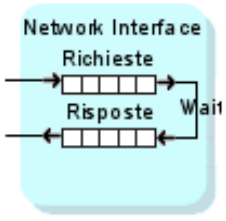
3.1.3 Richiesta-Risposta e Risposta-Richiesta

Questi tipi di dipendenze sono rappresentati dalle due catene di messaggi

dipendenza di richiesta-risposta
$m1_{richiesta} < m2_{risposta}$

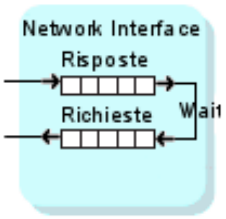
dipendenza di risposta-richiesta
$m1_{risposta} < m2_{richiesta}$

La prima delle due e' banale da capire: il fatto che una richiesta sia collegata alla sua risposta, e che questa puo' arrivare in momenti diversi, crea un fattore di attesa che, in presenza di determinate condizioni, puo' portare a deadlock.



Si nota che la rete e' assente da routing deadlock in quanto i buffer sono propriamente dimensionati e la rete e' aciclica. Vedremo come questo tipo di dipendenze si puo' eliminare dividendo i buffer in classi di richieste e risposte, in modo da

scollegare i buffer dei due tipi di messaggi, almeno logicamente. La seconda dipendenza, e' meno banale della prima:
puo' accadere che un nodo, una volta fatta una richiesta, e ottenuta la relativa risposta, reagisca producendo un ulteriore richiesta.In questo caso si possono generare questo tipo di dipendenze, che sono solitamente visibili in protocolli che sono simili ad un ciclo standard di $read < ack < write$ [37] o similari.



3.1.4 Richiesta-Richiesta e Risposta-Risposta

Questi tipi di dipendenze sono rappresentati dalle due catene di messaggi

dipendenza di richiesta-risposta
$m1_{richiesta} < m2_{richiesta}$

dipendenza di risposta-richiesta
$m1_{risposta} < m2_{risposta}$

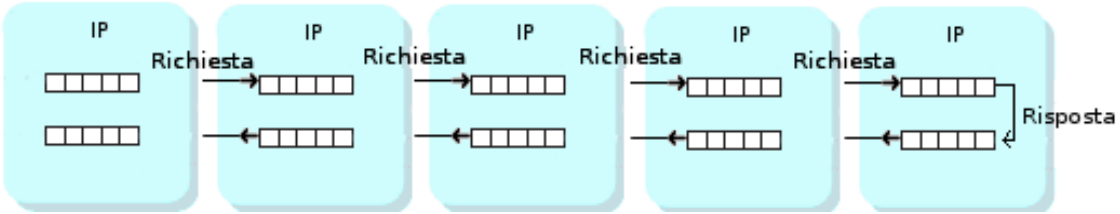
Queste dipendenze sono analizzate in un paper[7] che analizza gli effetti dell'introduzione di determinati protocolli nelle NoC. Queste dipendenze, in particolare, si generano quando, all'interno di un modulo, la ricezione di una richiesta da parte di un nodo destinatario, genera la produzione di un altra richiesta per un altro nodo.Quindi quello che succede e' che un modulo processa un certo input da un nodo che lo precede, e genera un input per il nodo successivo.Questo e' il caso tipico delle NoC dove viene usato il forwarding per aumentare le prestazioni, ma anche nei protocolli di tipo peer-to-peer, si pone questo problema.

Peer-to-Peer Streaming
Questo tipo di traffico,dove i dati vengono spinti attraverso la rete dal nodo sorgente al nodo destinatario, puo' causare dipendenze cicliche nella rete.
Un esempio di questi protocolli sono quelli che si occupano di fare encod-

ing/decoding video[24], dove i dati vengono processati per stadi successivi attraverso una pipeline.
Effettuare peer-to-peer con i dati, e quindi spingerli attraverso la rete, dimezza i tempi di latenza, in quando il nodo destinatario non deve necessariamente aver utilizzato del tempo per farne richiesta.
Questo tipo di traffico pero' genera dipendenze di tipo richiesta-richiesta.Infatti il traffico di tipo Peer-to-peer Streaming genera un grafo di dipendenze composto solamente dalle richieste in avanti del nodo successivo nella catena.

Forwarding

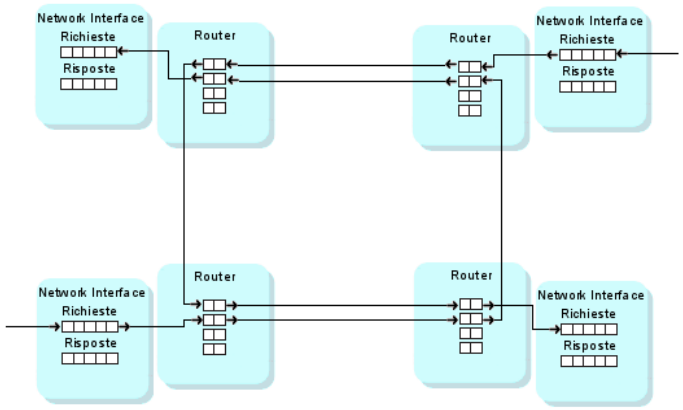
Il forwarding e' una tecnica utilizzata in alcuni protocolli per ottenere alcuni tipi di comunicazioni via messaggio.Tipicamente una richiesta iniziale passa attraverso un certo numero di nodi, in modo che la richiesta entrante generi nel nodo una nuova richiesta in uscita, fino a raggiungere la destinazione.In alcuni casi e' possibile che una richiesta in ingresso in un nodo da attraversare, generi un qualche tipo di acknowledgement verso il nodo che ha trattato precedentemente il messaggio.In questo caso si parla di dipendenze di risposta-risposta.Un tipico esempio che genera questo inconveniente sono quei protocolli che trattano multicast e narrowcast[8].



Come vediamo nell'immagine, la tecnica del forwarding genera una catena di richieste che vengono spinte attraverso la rete, vincolate ad un unica risposta.Questo meccanismo quindi puo' generare dipendenze all'interno della rete, causando potenzialmente il problema del mdd.
Solitamente questi protocolli sono utilizzati per gestire meccanismi di controllo di coerenza di dati su diverse memorie (come la cache) e tendono a voler minimizzare il numero di operazioni.Anche QuickPath di Intel utilizza un protocollo di multicast per il controllo di coerenza, e deve far fronte a questo tipo di dipendenze.Inoltre il multicast e il narrowcast possono essere utilizzati per implementare meccanismi di sincronizzazione e di acknowledgement e quindi rappresentano un potente mezzo del protocollo in una NoC per implementare funzionalita avanzate,nonostante possano dare origine a dipendenze di tipo richiesta-richiesta e

risposta-risposta

3.1.5 Routing deadlock



Il routing deadlock e' uno stato di blocco causato da una dipendenza non risolvibile all'interno dei buffer dei router appartenenti alla NoC.Nel caso del routing deadlock, il caso piu tipico e' la formazione di percorsi circolari all'interno del grafo delle dipendenze che rappresenta lo stato dei buffer, causati solitamente da un problema presente nel protocollo di routing.
Questo tipo di deadlock, infatti, si puo' risolvere tramite deadlock avoidance imponendo regole restrittive ai percorsi che i pacchetti possono compiere attraverso la rete. Un esempio di routing restrittivo,applicabile a una rete di tipo mesh,che impedisce il formarsi di percorsi ciclici e' il routing XY[30] che impone il tragitto dei pacchetti in questo modo:

1 - il pacchetto si muove lungo asse X fino a raggiungere la coordinata Y_d della destinazione
2 - il pacchetto si muove lungo asse Y fino a raggiungere la coordinata X_d della destinazione

Come si puo vedere, per risolvere il routing deadlock in maniera semplice, basta limitare la capacita' di movimento dei pacchetti,cosa che, in reti di dimensione contenute, non incide significativamente sulle prestazioni[6]. Esistono molti altri tipi di routing, deterministici, adattivi o parzialmente adattivi, che

riescono a risolvere il problema del routing deadlock, ma non verranno trattati in questo testo.
In ogni caso, risolvere il problema del routing deadlock, non incide sulla risoluzione del message-dependent-deadlock[6].

3.2 Gestione del deadlock

3.2.1 Deadlock Avoidance

La deadlock avoidance e' l'approccio piu semplice possibile all'eliminazione del deadlock.Solitamente ottenuto imponendo generiche restrizioni alla liberta' nella comunicazione (routing/messaggi), l'avoidance comporta sempre un prezzo da pagare, che sia una maggiore richiesta di risorse o un sottoutilizzo delle stesse. Ottenere l'eliminazione del deadlock e' poco costoso e molto rapido.Inoltre quando la rete e' composta da pochi nodi, gli approcci non adattivi, che limitano il routing, si dimostrano piu efficienti.

3.2.2 Deadlock Recovery

Se vediamo il deadlock come una situazione possibile, ma non probabile[6], esiste il modo di recuperare una situazione pericolosa, prima che giunga a compimento. Idealmente la deadlock avoidance elimina molti dei possibili partizionamenti di un sistema limitando di fatto la liberta di routing o di tipi di messaggi, perche, come si e' visto, e' la liberta di routing, o e' l'avere molti tipi diversi di messaggi, che massimizza il problema del deadlock.
E' possibile rimuovere un certo grado di liberta' solo quando una situazione di potenziale deadlock e' riconosciuta.Questo e' detto deadlock recovery. I deadlock in questo caso possono essere risolti togliendo i pacchetti dalla rete, e reinserendoli piu tardi in un dato momento(regressive recovery),convertendo, in un dato momento, un pacchetto da non terminante a terminante (deflective recovery),oppure possono essere risolti deviando forzatamente i pacchetti su percorsi garantiti (progressive recovery). Solitamente nell'approccio da supercalcolatori, viene usato un mix di recovery e avoidance.

3.2.3 Soluzioni al Message dependent Deadlock

Per garantire un assenza di deadlock, la condizione di consumo[1] deve essere soddisfatta.Assegnare un buffer diverso ad ogni tipo di messaggio, sulla NI, non

[1]La rete deve essere in grado di consumare quello che produce

e' sufficiente: cicli di routing e message-dependent-deadlock possono ancora causare stati di blocco[6].
La tecniche predominanti per risolvere questo problema, utilizzano solitamente la deadlock avoidance: un accurato dimensionamento dei buffer e un flow control che si occupa di non inserire nella rete piu di quanto la rete sia in grado di gestire, sono esempi per soddisfare la condizione di consumo. Altri possibili metodi per evitare questo problema riguardano l'uso dei Virtual Channel: dividendo logicamente la connessione in varie reti logiche separate, si possono dividere logicamente le code di dipendenza del tipo richiesta-risposta. I Virtual Channel poi, fanno parte di una tipologia di deadlock avoidance chiamata strict-ordering, dove a ogni tipo di messaggio e' associato un buffer separato, in modo da non creare dipendenze circolari tra i diversi tipi di messaggi. Vediamo i vari metodi nel dettaglio:

3.2.4 Dimensionamento dei buffer

Sovradimensionare i buffer e' una possibile soluzione al problema: si progetta la rete con un numero di buffer calcolato sul numero di tipi di messaggi possibili, in modo da garantire il consumo per ogni pacchetto immesso nella rete. Questo approccio e' poco utilizzato nelle NoC, in quanto comporta un grande spreco di risorse(i buffer sono solitamente sottoutilizzati).Sebbene sia un approccio spesso utilizzato nei supercomputer, non e' l'approccio favorito sulle NoC, dove spazio occupato,consumi e costi sono principi di progettazione molto importanti.

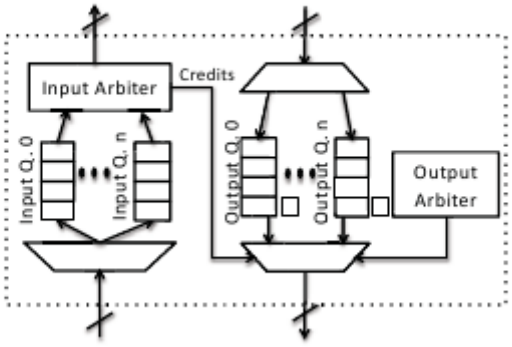
3.2.5 End-to-end flow control

Il flow control blocca la produzione di pacchetti quando la rete non e' piu in grado di garantirne il consumo.Solitamente basato su un sistema che tiene conto dei pacchetti inviati e ricevuti, il flow control, al contrario del sovradimensionare le risorse, come avviene nell'approccio precedente, utilizza al meglio possibile le risorse a sua disposizione.Un controllo di flow control end-to-end, poi, riesce a gestire tutti e 4 i tipi di dipendenze di messaggi illustrati sopra(richiesta-risposta e viceversa, risposta-risposta,richiesta-richiesta).Per i primi due il controllo avviene banalmente e localmente controllando che ci sia abbastanza spazio disponibile sul percorso, per gli altri due e' necessario un meccanismo basato su crediti (credit based flow control), che permette di controllare lo stato della rete anche con l'utilizzo di protocolli di comunicazione che presentano streaming o forwarding.
L'end-to-end flow control ha l'indubbio vantaggio di non dover conoscere, per la

gestione dei deadlock, i tipi di messaggi inviati.In questo modo i router sono piu semplici,in quanto servono contatori di messaggi non tipizzati. D'altra parte il meccanismo dei crediti necessita di buffer dedicati, per tenere traccia dei crediti, necessita di canali di comunicazione dedicati (virtuali o non), che comunque consumano bandwidth.

CB, Credit-Based flow-control

Questo tipo di approccio e' implementato nel nodo di rete tramite una serie di code, all'interno della NI(al contrario delle VN, dove le code devono essere replicate sia sulle NI che all'interno dei router), che si occupano di gesire lo scambio di crediti,che non sono altro che contatori, sia in input che in output.

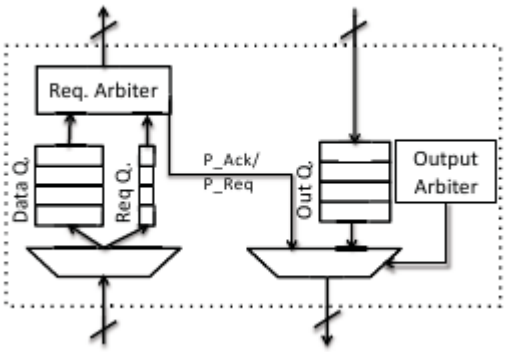


L'immagine rappresenta una normale NI con i buffer necessari al suo funzionamento tramite CB flow-control. Ogni nodo che nella rete vuole inviare pacchetti, tiene traccia dello stato delle code di ricezione del nodo destinatario, e invia quando e' garantita la disponibilita all'interno dei buffer.I nodi, per ternersi aggiornati riguardo allo stato delle code, si scambiano dei messaggi contenuti in altri pacchetti, oppure utilizzano dei pacchetti dedicati per gestire questo tipo di comunicazioni.

CTC,Connection-then-Credit flow-control

Questo meccanismo di flow-control e' stato presentato di recente[34] come alternativa al CB per reti best-effort.E' basato sul flow-control Credit-Based, ma si distingue per una diversa architettura all'interno della NI.
Al contrario del CB flow control, che richiede un numero di code dipendente dalla topologia della rete,il CTC flow-control richiede due code per i dati in input,e una singola coda per quelli in output. Una delle due code in input e'

utilizzata per i dati in arrivo dalla rete, mentre l'altra e' utilizzata per gestire le richieste di connessione pendenti.



L'immagine rappresenta una NI semplificata utilizzando il flow control di tipo ctc.Come si puo vedere, l'architettura viene di molto semplificata vista la minore richiesta di buffer di questo algoritmo di flow-control
Ecco sinteticamente la procedura di comunicazione per un nodo A che ha necessita' di comunicazione verso un nodo B:

- il nodo A genera una richiesta di tipo P_{REQ}, che verra' inserita nell'apposita coda dal nodo B
- il nodo B, aspetta la gestione di nuove richieste,che avverra' nel momento in cui il buffer dati in ingresso rendera' possibile una nuova comunicazione
- il nodo B, una volta scelto (in base a una politica FIFO, o dipendente da una priorita') il prossimo nodo da cui dovra' ricevere, invia a quel nodo un messaggio di tipo P_{ACK}
- il nodo A inizia la comunicazione.

E' importante notare come i pacchetti P_{REQ}/P_{ACK} contengano informazioni aggiuntive per le NI, che gli permettono di gestire eventualmente diversi tipi di priorita, e un inizializzazione precisa dei contatori dei crediti, in modo che l'interfaccia che ha generato il P_{ACK} riesca ad "attirare" verso di se il giusto quantitativo di dati,tramite delle catene di pacchetti P_{ACK}.
I punti forti di questo meccanismo risiedono nella sua capacita' di semplificare l'architettura della NI, riducendo il numero di buffer richiesti e rendendoli indipendenti dall'architettura(tre code sono richieste in totale, a prescindere dal numero di nodi).
Il trade-off, in questo caso, risiede nel fatto che ogni nuova comunicazione deve

essere preceduta da una procedura di handshake, cosa che aumenta la latenza complessiva della NoC, e riduce la banda. Per alcuni tipi di applicazioni, quali lo streaming video, dove una serie di piccoli pacchetti devono passare una serie di stadi in pipeline, per esempio, il costo di latenza comunque rimane sotto al 10%, mentre lo spazio risparmiato, dato dalla semplificazione delle NI, raggiunge il 35%, rendendo il CTC una soluzione risparmiosa ed efficiente.

3.2.6 Virtual Networks

Il virtual networking rappresenta una soluzione per risolvere il problema del deadlock: i diversi tipi di messaggi e le connessioni possono venire logicamente separati in modo da non creare cicli. Anche in questo caso, come nei due approcci precedenti, la risoluzione del deadlock e' rappresentata da un consumo di risorse: i buffer (logicamente separati tra loro), vengono generalmente sottoutilizzati.Inoltre in questo approccio, il numero massimo di canali virtuali e' dato dal massimo numero di buffer disponibili nella NI, deciso in fase di progettazione.

3.2.7 Strict Ordering

In questo approccio, che fondamentalmente e' un sovra-insieme della soluzione basata sulle Virtual Networks, vengono creati buffer e reti logicamente(o fisicamente) separati.Il principio fondamentale e' dedicare un determinato buffer a un determinato tipo di messaggio, in modo da spezzare, in fase di progetto, le dipendenze tra i tipi di messaggi.Si tratta di una soluzione meno dinamica delle Virtual Networks, in quanto un determinato buffer viene legato a un certo tipo, e non viene permessa la condivisione tra risorse.In questo modo vengono solitamente richieste maggiori risorse (che vengono poi sottoutilizzate), e la congesione della rete e' solitamente maggiore, per via dello sbilanciamento tra i vari buffer.Sempre negli approcci provenienti dal mondo dei supercomputer, vengono proposti approcci[6] quali il diverso dimensionamento dei vari tipi di buffer(sempre pero' a livello di progettazione), che mitigano lo spreco di risorse sulla rete, ma questo genera squilibri nel traffico trasmesso.Con questo approccio, i protocolli che usano forwarding, streaming, o tecniche higher ordine, devono essere ridotti a protocolli richiesta-risposta o risposta-richiesta, in quanto gli altri due casi (risposta-risposta e richiesta-richiesta) non vengono gestiti dallo strict ordering.La riduzione del protocollo naturalmente introduce poi la richiesta di maggiore bandwith e un maggior numero di buffer.In generale,

comunque, questa soluzione si dimostra poco flessibile e meno utilizzata rispetto alle due precedenti.

3.2.8 Valutazione dei vari approcci

Come mostrato, esistono vari approcci possibili per risolvere il deadlock, e anche se le varie NoC ne usano principalmente 2-3 tipi, e' interessante vederne la diffusione.

3.2.9 xpipes

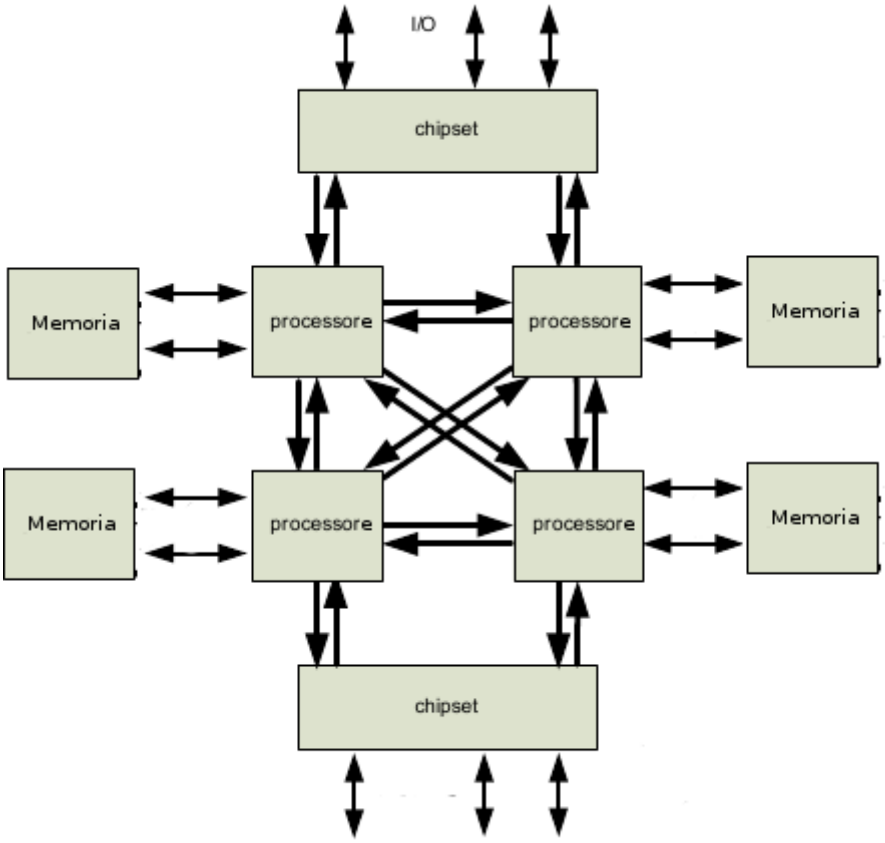
XPipes[31][32] rappresenta un progetto dell'universita' di Stanford e di Bologna, in collaborazione con STMicroelectronics, che l'ha abbandonata per concentrarsi su "progetti futuri" basandosi su quello che ha ricavato dagli studi di questa architettura.Xpipes e' una NoC che, sebbene preceda molte delle NoC "attuali" ne presenta tutte le caratteristiche peculiari, e questo ne rende interessante l'analisi. E' composta da una serie di link tra i vari nodi, una serie di switch[2], e una serie di Network Interface.
Le Network Interface convertono i segnali (basati sull'OCP[3]) in flit che vengono smistati per la rete.Il protocollo su cui si appoggiano le NI e' composto solamente da istruzioni di tipo 'richiesta-risposta'.
Gli switch mettono in pratica la funzione di routing della rete, che si basa su routing statico, e supportano wormhole routing.
Questa architettura e' stata progettata sviluppando un set di librerie che permettono, tramite un simulatore, di testarla.Sfortunatamente e' stata abbandonata in previsione di futuri sviluppi,sempre in collaborazione con STMicroelectronics, su piattaforme di tipo NoC, e quindi a tutt'oggi si tratta di un architettura completamente definita, ma che non supporta alcun sistema per prevenire un protocol deadlock.

3.2.10 STBus

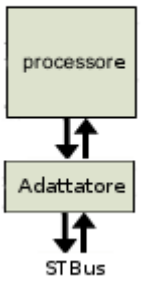
STBus[27][35] e' un progetto STMicroelectronics in collaborazione con l'universita' di Stanford, che realizza una soluzione intermedia tra il bus condiviso standard e un approccio di tipo NoC.
STBus puo' essere istanziato, in un architettura, in tre modalita: bus condiviso,crossbar parziale,crossbar totale.

[2] uno per ogni nodo, con funzionalita' paragonabili a quelle di un router nell'accezione delle NoC

[3] Open Core Protocol



Mentre il primo caso e' equivalente alle architetture standard basate su bus, gli ultimi due rappresentano una serie di interconnessioni tra i componenti capaci di operare in parallelo.
Il caso dell'immagine sopra, e' di un STBus parzialmente connesso con vari processori e varie memorie.
E' da notare che questa architettura e' stata inizialmente progettata per supportare nodi di diverso tipo, ed essere immediatamente implementata su architetture reali, con componenti potenzialmente molto diversi che possono essere interconnessi tra loro da bus di diverse dimensioni.
Per questo motivo STBus si presenta come una soluzione che non stravolge l'originale approccio basato sul bus condiviso, ma presenta le stesse problematiche di una NoC(come, ad esempio, il deadlock), sebbene non abbia componenti simili. La rete, che e' svincolata da qualsiasi tipo di topologia, ma viene progettata in base alle esigenze che si vogliono raggiungere,in termini di banda e latenza, e' composta dai vari nodi(che nient'altro sono se non i singoli componenti), e da un certo numero di adattatori.



Vi e' almeno un adattatore per bus, ma un adattatore puo' servire a piu nodi per le comunicazioni, in base alla progettazione.Quello che fa' l'adattatore e' rendere compatibile il traffico tra i vari nodi, che puo' essere di tipo diverso(per esempio far convivere unita di calcolo, potenzialmente diverse, con vari componenti di memoria). In modalita' full crossbar la rete e' completamente interconnessa, garantendo la massima banda e la minima latenza, ma questa soluzione e' sconsigliabile per via di costi,consumi e volume occupati. Inoltre non e' garantito che una full crossbar sia piu performante di una crossbar parziale opportunamente dimensionata. Esistono studi[17] che rendono, tramite delle simulazioni, una crossbar parziale simile in prestazioni a una full crossbar, anche se chiaramente questo puo' essere fatto riferendosi a un caso specifico di architettura, e non puo' essere un caso generale.

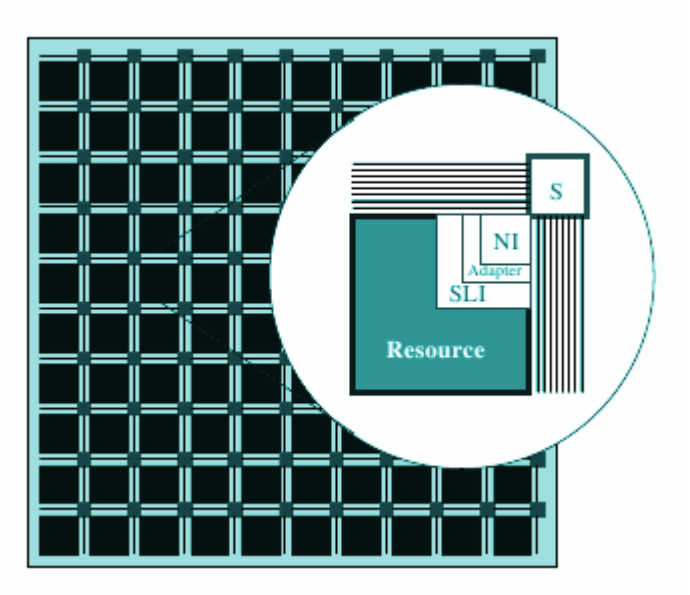
Sebbene non si possa parlare, nel caso di STBus, di un vero e proprio protocollo di tipo message-passing, la rete effettua tra i vari componenti una serie di chiamate di tipo "richiesta-risposta", ed e' per questo potenzialmente soggetta al protocol deadlock.

In questo caso la soluzione adottata e' lo strict-ordering. Questa soluzione consiste nella divisione fisica di due reti: una dedicata alle richieste e l'altra alle risposte.In questo caso questa soluzione, che e' statica, e non consente di implementare un protocollo a piu di due tipi di messaggio, e' sufficiente per eliminare il rischio di protocol deadlock.

Al contrario delle virtual network, i canali qui sono assegnati staticamente, e non possono essere scambiati a seconda delle esigenze. In questo modo la rete e' dimensionata staticamente in base alle esigenze dell'architettura che si vuole andare a ottenere. Chiaramente si tratta di una soluzione molto rigida, ma e' sufficiente in questo tipo di architettura.

3.2.11 Nostrum

Nostrum[16] e' una NoC sviluppata a Stoccolma, da un gruppo di ricercatori del Royal Institute of Technology, con l'obiettivo di essere una NoC il piu' possibile somigliante come stack al TCP-IP standard in uso nelle reti.



Per questo motivo presenta uno stack a livelli molto somigliante a quello in uso nelle grandi reti: 5 sono i livelli presenti, fisico,data link,rete,sessione,applicazione. Nostrum presenta una topologia standard mesh, scelta per motivi di predicibilita' dei ritardi,omogeneita' nella progettazione dei vari nodi e per motivi di localita' del traffico[4]. Un nodo della rete e' composto da un unita' di calcolo che e' interconnessa alle unita' adiacenti tramite uno switch[5].
La descrizione dei vari livelli e' molto lineare, e si parte dal livello fisico, che rappresenta le linee di interconnessione che vi sono tra nodo e nodo.

Il livello data-link e' composto dallo switch, che, come uno switch di una rete LAN, si occupa di instradare il traffico dei livelli superiori verso la giusta destinazione. Quest'ultimo e' capace di effettuare controlli di correttezza sulle comunicazioni, ed e' dotato di buffer, per immagazzinare i flit in ingresso/uscita.

Il livello di rete,confrontandolo con i componenti di una generica NoC, assume parzialmente le funzionalita' di Router e NI: questo infatti implementa il protocollo di routing, e, tramite il meccanismo dei virtual channel, si occupa della

[4]Un nodo in questa rete tendenzialmente comunichera' piu spesso con nodi adiacenti
[5]Non paragonabile al router di una generica NoC

multiplazione dei canali su certi tipi di comunicazione garantita.E' da notare che, fino al livello di rete, non si parla di messaggi o pacchetti, ma a questo livelli si trattano ancora i singoli flit.

Il message-passing entra in gioco al livello sessione, dove viene implementata un interfaccia per i livelli superiori basata su message passing e un set di primitive di rete basilari (lettura/scrittura,apertura-chiusura canali,ecc.).

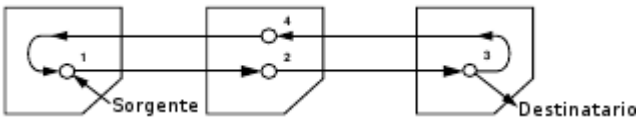
Il livello applicazione fornisce, cosi come la sua controparte implementata nelle reti, un set di servizi specifici e differenziati per applicazione che sulla NoC sta eseguendo in un dato momento.
Nostrum, come MANGO o AEthereal, fornisce un doppio tipo di traffico utilizzabile in trasmissione: best-effort o garantito.
Nel primo caso del messaggio e' garantito semplicemente l'arrivo, non curandosi dei ritardi.Nel secondo caso viene garantito l'arrivo, una certa quantita di banda e un delay massimo per la ricezione dell'intero messaggio.
Nostrum ha tra i suoi obiettivi il mantenere molto basso l'impiego di hardware, e per questo adotta alcune soluzioni uniche nella sua gestione. Per raggiungere questi obiettivi in termini di costi,disturbi sulla rete e consumi, Nostrum ha scelto una politica di switching che e' il deflective routing che, rispetto ad esempio ad altre reti (come AEthereal), non richiede la presenza di tabelle di routing e code in ingresso/uscita per i pacchetti in trasmissione sulla rete. Il traffico best effort e' gestito sepmlicemente dall'invio di datagram contenenti i dati, sulla rete.Le decisioni sono fatte localmente allo switch in cui il pacchetto e' in transito su base non deterministica per ogni datagram che transita sulla rete (i datagram che compongono un pacchetto possono percorrere percorsi differenti per giungere a destinazione).
Nostrum raggiunge l'obiettivo della banda garantita (GB) attraverso l'uso dei Virtual Circuits, componenti implementati tramite l'uso di due altri oggetti, chiamati dagli sviluppatori Looped containers e Temporally disjoint networks.Questi componenti sono molto economici,sia in termini di implementazione,che in termini di banda usata per garantire il servizio, anche se, come vedremo, non sono privi da difetti.

Per implementare la GB, Nostrum usa un particolare tipo di messaggio, detto "container".Questo differisce dai datagram standard per due motivi:
-segue un percorso predefinito
-puo' essere privo di contenuto

Come detto,Nostrum utilizza un particolare tipo di routing detto Deflective Routing[33].Questo implica il fatto che non vi sono code con la funzione di riordino dei pacchetti,e non vi e' un invio che costringa l'invio e l'arrivo dei datagram in uno stesso ordine.Questo perche' i datagram che compongono un pacchetto possono prendere strade diverse sulla rete, e le decisioni sul routing sono fatte localmente e dinamicamente e quindi la lunghezza del percorso che un pacchetto deve fare, puo variare.

Looped Containers

Il looped container e' un meccanismo che permette alla rete Nostrum di occupare sicuramente un canale di comunicazione, in un dato momento temporale. E' dato che all'interno degli switch, un pacchetto che attende per essere immesso nella rete, non puo' prendere possesso di un canale, se questo e' occupato.Se tutti i canali sono occupati, il pacchetto in attesa dovra' aspettare fino a che uno dei canali non venga liberato.

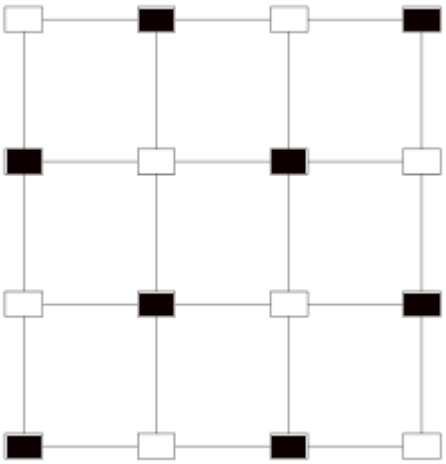


Il funzionamento del Looped container,come mostrato nella figura qui sopra, permette a un pacchetto di prendere in un dato momento possesso del canale;il suo funzionamento e' molto simile a quello dello standard Token Ring[36]: un pacchetto gira indefinitamente all'interno di un canale tra due nodi, anche se i nodi non sono attualmente in fase di comunicazione.
Nel momento in cui uno dei due nodi vuole trasmettere un pacchetto all'altro, si impossessa del Looped Container,che altro non e' che un datagram vuoto, e lo riempie con i dati che devono essere trasmessi con GB.Il pacchetto riempito di dati viene trasmesso fino alla destinazione, viene svuotato e rimandato indietro, facoltativamente con informazioni aggiuntive del nodo destinatario(messaggi di tipo ACK o altro),e il ciclo puo' ricominciare.

Temporally Disjoint Networks

Questo meccanismo si basa sul fatto che pacchetti emessi nello stesso ciclo di clock possono collidere solamente se la loro distanza non e' un multiplo del singolo percorso di round-trip tra due nodi.Per questo motivo se dividiamo la rete in due tipologie di nodi, avremo l'insieme dei nodi partizionato in due parti che

contengono lo stesso numero di nodi.



Chiaramente un pacchetto puo passare indifferentemente da nodi di un tipo all'altro,mostrati nella figura come bianchi o neri, ma e' sicuro che, dati due pacchetti su diverse partizioni in un dato momento, siamo sicuri che non e' possibile che questi due si incontrino(cioe' siano sullo stesso segmento di rete tra due nodi).

L'architettura Nostrum, quindi, presenta un gran numero di livelli che ne aumentano la modularita', e implementa funzionalita' inusuali di cui altre NoC sono prive (protezione dei dati multilivello, un simulatore dedicato, uno switch che puo' essere implementato privo di buffer). Questo sebbene abbia costrizioni di topologia, e un livello di rete molto complesso (che lascia invece un livello data-link molto semplice e, in una sua particolare implementazione, privo di buffer), se paragonato a quello di routing e/o NI di una comune NoC.
Cosi come MANGO, Nostrum, nella sua implementazione di traffico best-effort, non offre alcuna protezione riguardo al protocol deadlock, e questo problema viene semplicemente ignorato Al contrario, utilizzando traffico garantito, Nostrum permette di evitare routing deadlock e protocol deadlock tramite l'utilizzo dei virtual channel e quindi delle virtual networks.
Purtroppo il meccanismo dei Looped Containers porta anche un significativo svantaggio: per il loro funzionamento che si basa sul partizionare la rete in due possibili canali che devono essere sincronizzati (vista anche una potenziale assenza ddi buffer negli switch) per mantenersi privi di collisioni, si deve utilizzare

un quantitativo di banda equivalente in entrambe le direzioni, introducento uno spreco significativo.
Inoltre i protocolli richiesta-risposta devono fare transitare le risposte nel canale sul quale non transitano le richieste, e quindi non si possono implementare protocolli che non siano di tipo master-slave, a meno che non si prendano particolari precauzioni nell'utilizzo dei buffer.

3.2.12 AEthereal

La rete AEthereal,sviluppata da Philips, e' una NoC sincrona composta da due tipi di componenti, router e NI.

Router

I router delle rete AEthereal si occupano dell'end to end da router a router,supportano wormhole routing ed eseguono controlli di correttezza su i flit in arrivo. Non vi sono vincoli di topologia,nella rete, e possono esservi piu link tra due router.
Il routing e' basato su indirizzi, e questo permette di essere indipendenti dalla topologia.Nell'header di un pacchetto, infatti, vi e' la lista del percorso che dovra fare, e a ogni passaggio in un nodo, verra' fatta un estrazione dalla cima della lista fino a giungere a destinazione. Nel routing per garantire il throughput, i pacchetti GT sono schedulati per essere processati al successivo ciclo mentre BT sono schedulati secondo un ciclo round robin. I router usano il wormhole per essere piu economici nei buffer(non tutto il pacchetto deve esser contenuto nel buffer per inviare, e quindi si riducono le dimensioni dei buffer), e nella latenza (si puo forwardare senza aspettare l'intero pacchetto).

Network Interface

Le network interface della rete AEthereal si occupano della pacchettizzazione dei messaggi, dell'end-to-end della comunicazione da nodo a nodo,dell'ordinamento dei messaggi e presentano un flow control end-to-end.. Le NI sono divise poi in due sezioni: kernel che si occupa della pacchettizzazione e dello schedulare i router, e shell, che offrono servizi per le connessioni (quali il multicast) agli strati superiori.
Inoltre vi sono shell, sulla NI, che provvedono ad altre funzionalita, quali il multicast e il narrowcast, per gli strati superiori, e hanno adattatori per i protocolli in uso quali AXI o DTL. Ogni shell, in ogni nodo, puo essere connessa o no, in ordine di contenimento costi e spazio.

Traffico e flow control
AEthereal usa servizi configurabili per dare garanzie di throughput e latenza, e possono essere selezionati a piacimento sulla noc: in questo modo puo' offrire sia servizi garantiti che best-effort a seconda delle esigenze. La rete supporta due tipi di traffico:BT(best effort throughput) e GT(guaranteed throughput).
Il flow-control e' utilizzato per fare in modo che i buffer non vadano in overflow, causando perdite di pacchetti.Quindi ogni router che accetta un pacchetto, segnala il suo spazio decrementato alla rete. Quindi, mentre il traffico di tipo BT segue una coda, il traffico GT segue un loro circuito pipelined, che consente in ogni caso, al successivo ciclo, di passare al nodo successivo. Il flow control tiene traccia dell'occupazione dei vari buffer tramite un sistema di crediti e l'intera rete si basa su un sistema sincrono che permette a tutti i nodi di seguire uno stretto time-division multiplexing,senza il quale non sarebbe possibile creare una serie di servizi garantiti quali il GT.

Questa rete,quindi, offre servizi high-level, wormhole routing,flow control, garanzie su throughput e latenza, ed e' semplice essendo composta solamente da router e NI. I router hanno una coda in ingresso, e supportano due classi (garantita e best effort) per il traffico.Per la garantita non ce contesa, ogni pacchetto e' forwardato sempre su un canale privilegiato.Per il best effort viene utilizzato il round robin per selezionare il pacchetto successivo da inviare.Ogni router non conserva le slot table che configurano il nodo, questo per privilegiare spazio e costi, visto che la rete e' solitamente composta da pochi nodi. Tutte le connessioni di rete sono configurabili a runtime attraverso una porta mappata in memoria.

3.2.13 Mango

Mango[14] e' una NoC asincrona, pensata per essere implementata su NoC estese(almeno 25 nodi,nelle simulazioni).Crescendo di dimensione, infatti, la gestione di un clock sincrono diventa una sfida di progettazione, e Mango vuole ovviare a questo problema proponendosi come una rete asincrona che, come AEthereal, puo' fornire garanzie da un punto di vista di bandwidth e latenza. Quindi Mango adotta un approccio di tipo GALS, Global Asynchronous Locally Synchronous, cioe' un set di nodi internamente sincroni comunicano tra loro in modo asincrono. Mango, inoltre, adotta per l'implementazione dei nodi, lo

standard OCP[29], che fornisce una serie di standard e di interfacce flessibili per la progettazione di nodi di una NoC, basati su message passing.
Gli sviluppatori di Mango, quindi, puntano su questa rete fornendone, come punti di forza, la scalabilita', la modularita', l'adesione a standard quale OCP e la bassa latenza dei pacchetti in pipelining, dovuta al fatto che la rete e' asincrona. Il sottosistema di comunicazione di MANGO e' composto da Router, Link e Network Adapters[6]. Le Network Adapters si occupano di fornire i servizi ad alto livello per gli strati superiori, quali operazioni di scrittura e lettura attraverso la NoC, e inoltre si occupano di traslare le comunicazioni in ingresso o in uscita dall'unita' di calcolo del nodo (sincrona) attraverso la rete (asincrona). I Link non sono altro che le interconnessioni che vi sono tra nodo e nodo, e, in definitiva, rappresentano il numero di porte attraverso il quale il router puo' comunicare. I router[15] della rete MANGO hanno una doppia modalita', a seconda che la connessione debba avere servizi garantiti o best-effort.Infatti nel caso di un servizio best-effort, la connessione deve essere di tipologia connectionless(in modo molto simile a un protocollo di tipo UDP), mentre nel caso di un servizio garantito, il protocollo viene orientato alla connessione con il nodo di destinazione. La rete MANGO utilizza il meccanismo dei Virtual Channels, per multiplare i canali disponibili e risolvere i problemi di deadlock a livello routing, e la gestione di questi virtual channels e' lasciata a un modulo (detto VC module), che si occupa di effettuare un primitivo flow control sui canali.Il protocol deadlock, allo stesso modo, e' gestito quindi tramite la separazione logica dei canali in trasmissione su diversi tipi di messaggi del protocollo.La gestione e' affidata al nodo, e l'utilizzo dei vari virtual channel viene "dosato" in modo da non mandare all'interno dello stesso canale messaggi che possono generare delle dipendenze.

3.2.14 Intel QuickPath

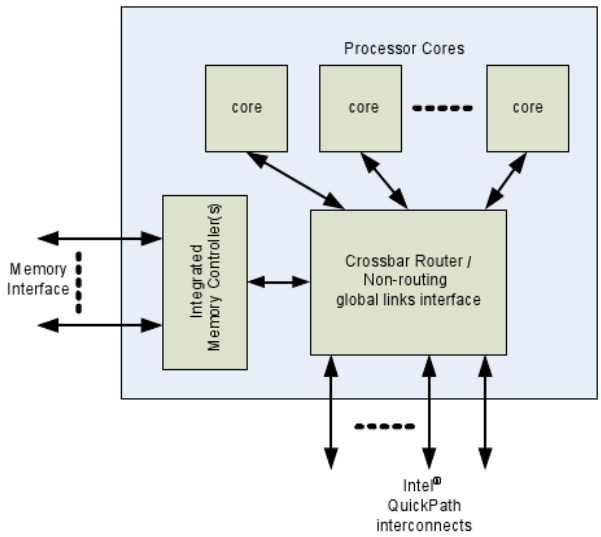
Questa archiettura e' molto importante, in quanto rappresenta la proposta Intel, leader nel mondo dei microprocessori per personal computer, per integrare le NoC in un contesto di diffusione su larga scala. QuickPath e' gia operativo su microprocessori di fascia alta(serie Xeon) addirittura dalla seconda meta del 2008.Quello che fa', e' sostituire il vecchio bus condiviso con una serie di link punto a punto ad alta velocita tra processori, e da processore verso la memoria.Tutti i trasferimenti tra link separati possono avvenire contemporaneamente, cosi come in una NoC. Unendo questo al fatto che QuickPath presenta tutte le caratteristiche e le problematiche di una normale NoC, si puo' dire che questa

[6] questi sono paragonabili alle Network Interface, nell'accezione standard delle NoC

architettura rappresenti una Netowrk on Chip, anche se Intel non si riferisce ad essa come tale.

Il nodo

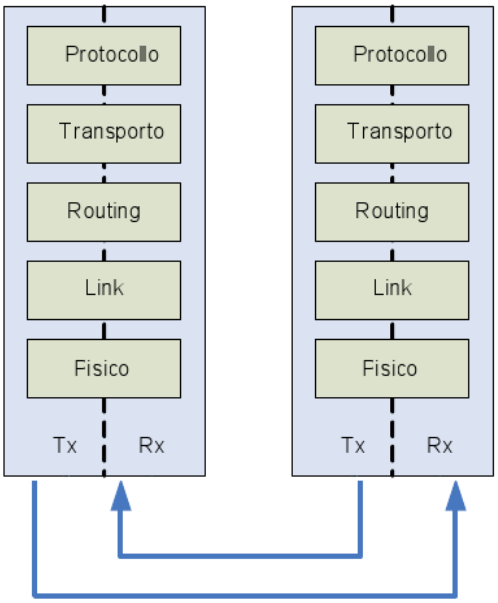
In un architettura QuickPath(QP),la rete e' formata da nodi, tipicamente processori o controller di memoria.Vediamo un esempio di nodo-processore:



Come si puo vedere, un processore e' previsto che possa essere anche multi-core, e l'apparato che Intel chiama crossbar router altro non e' che un router che gestisce sia le comunicazioni interne tra i vari core, che le comunicazioni esterne con gli altri nodi [9].E' da notare che i core possono (ma non e' necessario) avere una propria memoria cache privata, e il router in questo caso si occupa di gestire anche la coerenza delle informazioni tra varie cache.
Come Intel dimostra, la gestione della coerenza tra le varie cache e' effettuata inviando particolari tipi di messaggi ai vari nodi della rete.

Lo stack

QP ha un architettura equivalente a quella di tutte le altre NoC in commercio.Si basa su uno stack formato da 5 livelli: Fisico,Link,Routing,Trasporto,Protocollo.



Livello Fisico:si occupa di trasferire fisicamente i segnali.L'unita' di misura e' il phit (physical-unit) di 20 bit, che e' l'equivalente fisico di cio' che sulle normali NoC e' detto flit.I phit, nelle architetture QP, sono sottomultipli dei flit utilizzati a livello di link.

In generale il livello fisico e' quindi composto da una serie di coppie di link unidirezionali, una coppia per collegamento.

Livello Link:si occupa del flow-control e di fornire comunicazioni affidabili.L'unita di trasferimento e' il flit (flow control-unit) composto da 80 bit.
che di controllare la coerenza tra i dati nelle varie memorie, in modo che vengano sempre tenuti aggiornati, ma di questo non tratteremo.

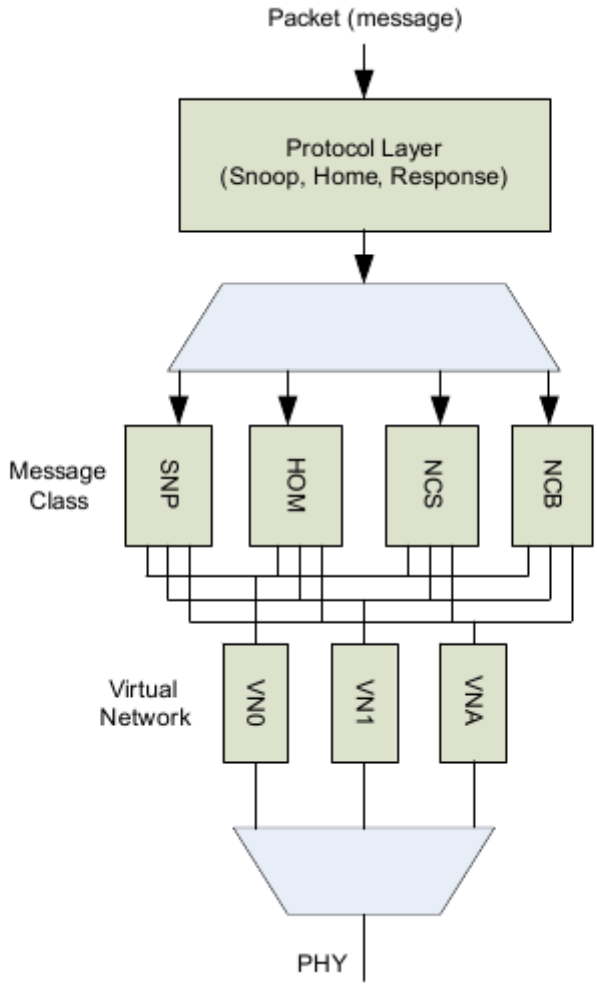
La gestione del deadlock
Intel QP utilizza come architettura due meccanismi di deadlock avoidance per risolvere il problema del routing deadlock e del protocol deadlock.

- Message Classes
- Virtual Networks

Il livello di Link supporta sino a 14 classi di messaggi di cui 6 sono attualmente specificate, e le altre 8 sono attualmente indefinite per sviluppi futuri.Un messaggio in arrivo dal livello Protocollo viene smistato in una delle 6 classi in

uso.In questo modo diversi messaggi in uscita dal nodo vengono multiplexati su buffer logicamente scollegati, tramite questo meccanismo che e' equivalente a quello dei virtual channel.

Le classi di messaggi attualmente in uso sono le seguenti:
SNP,NDR,DRS,NCS,NCB,HOM.
La classe HOM necessita, per motivi di protocollo, di riordinamento dei flit all'interno del buffer,sia in invio che in ricezione, mentre le altre 5 classi possono spedire e ricevere flit non necessariamente in maniera ordinata.
Le virtual networks sono un un modo aggiuntivo che QP utilizza per smistare il traffico in maniera ancora piu efficiente, replicando ogni classe di messaggi su diversi canali potenzialmente separati tra loro.Sono state introdotte per supportare una serie di funzionalita piu avanzate della rete, quali routing affidabile,indipendenza dalla topologia e riduzione dei buffer attraverso un uso sapiente dei buffer che le virtual network offrono.
Le virtual networkS suddividono le classi di messaggi in partizioni, in modo che diversi set di messaggi possano transitare su reti logicamente scollegate.Tre virtual network sono supportate attualmente, e sono chiamate VN0,VN1 e VNA.Le prime due sono indipendenti tra loro,e creano due reti logicamente scollegate,mentre la terza ha un buffer che puo' essere condiviso tra set di messaggi che non collidono tra loro, per un uso generico.Chiaramente il numero totale di connessioni possibili e' dato dal prodotto delle message classes, che attualmente sono 6, per il numero di virtual networks, in modo che si possano creare, per ogni virtual network, 3 virtual channel per ogni classe di messaggi (3 SNP,3 NDR, ecc..),per un totale di 18 virtual channel.



Oltre al meccanismo combinato di message classes e virtual networks, QP utilizza un sistema di flow control semplice basato su crediti.A ogni nodo della rete viene assegnato un certo numero di crediti,spendibili su una coppia[N,B], dove N rappresenta un nodo di destinazione e B rappresenta un buffer.Ogni volta che un flit viene inviato su un determinato buffer, i crediti che un mittente ha per quel nodo e quel buffer vengono decrementati di uno. Se il contatore raggiunge lo zero, il mittente blocca l'invio di altri pacchetti sulla rete, mentre ogni volta che un pacchetto viene consumato dal destinatario, il credito corrispondente viene ri-aumentato.Le informazioni riguardo ai crediti viaggiano sulla rete all'interno dei flit contenenti i dati, e non vi sono messaggi appositi che si occupano di questa precisa funzione.
Quindi QP utilizza un meccanismo di deadlock avoidance basato sulle virtual

networks e il partizionamento logico del set di messaggi, per impedire sia il routing deadlock che il protocol deadlock, insieme a un meccanismo di flow control che impedisce la congestione della rete.

3.2.15 Valutazione dei meccanismi di Deadlock Avoidance

Come abbiamo visto, nelle reti Mango e Nostrum con servizi best-effort, e xpipes, le dipendenze che possono creare un protocol deadlock non sono gestite, e quindi queste reti sono soggette a protocol deadlock.Infatti in questi tre casi nemmeno un protocollo basato su due tipi di messaggi (richiesta-risposta) puo' essere utilizzato con sicurezza.

NoC	Strict Ordering	Virtual Networks	Flow Control
XPipes	.	.	.
Mango BE	.	.	.
Mango GS	.	X	.
Nostrum BE	.	.	.
Nostrum GS	.	X	.
AEthereal BE	.	.	X
AEthereal GS	.	X	.
Intel Quickpath	.	X	.
STBus	X	.	.

Al contrario, i servizi garantiti di Mango, Nostrum,Quickpath e AEthereal, gestiscono il protocol deadlock tramite il meccanismo delle Virtual Networks, anche se questo implica un possibile spreco di risorse e, nel caso di Mango e Nostrum, un numero massimo fissato di connessioni.
STBus costituisce un eccezione, e gestisce con successo il protocol deadlock tramite il meccanismo dello strict ordering e della divisione delle reti di richiesta e risposta in due classi separate.Questo pero' porta ad avere una NoC che non puo' gestire piu di due tipi di messaggi(richiesta-risposta), e non puo' nemmeno gestire protocolli piu complessi che implementano forwarding o streaming. I servizi best-effort di AEthereal utilizzano un meccanismo di crediti e gestione del flow control complesso, che permette di soddisfare la condizione di consumo della rete[7]. In questo caso la rete e' indipendente dal protocollo, permettendo alla rete AEthereal di non introdurre dipendenze tra messaggi qualsiasi sia il protocollo implementato, e il numero massimo di connessioni possibili e' dipen-

[7]la rete e' in grado sempre di consumare quello che produce

dente semplicemente dalla progettazione e il dimensionamento dei buffer delle NI.

3.2.16 Valutazione dei costi e dei consumi

Apparentemente la soluzione piu utilizzata(e forse piu flessibile), cioe' l'utilizzo di virtual networks, e' anche la piu onerosa in termini di costi, consumi e volumi occupati, come studiato in alcuni paper[26][27] che analizzano i consumi e i trade-offs nella progettazione di queste architetture.
Secondo questi studi[7], le soluzioni migliori in termini di numero di componenti, costi, consumi e volumi occupati, sono quelle basate su strict-ordering e end-to-end flow control.
Prendendo l'architettura d'esempio proposta in[24] e modellandola tramite UMARS[28],si nota che lo strict-ordering occupa uno spazio nell'architettura strettamente inferiore al 3.9% mentre il controllo basato su flow control riesce ad essere contenuto fino allo 0.3% del volume totale.

3.2.17 Conclusioni

Come abbiamo visto il message-dependent-deadlock e' un problema serio che puo' minacciare il corretto funzionamento di una NoC.
I modi per risolvere questo problema sono molteplici, e risiedono nella maggioranza dei casi nella deadlock avoidance che, tramite l'eliminazione delle dipendenze circolari, impedisce il verificarsi di condizioni di blocco.
I meccanismi ottimali, da un punto di vista di costi di implementazione,consumi, volumi occupati e prestazioni, sono quelli basati su un flow control end-to-end che, supportando servizi avanzati, puo' gestire il deadlock anche su protocolli n-way dotati di funzionalita quali il forwarding,il peer-to-peer streaming,il narrowcast o il multicast, ottimizzando l'utilizzo delle risorse.
I sistemi di flow-control credit based sono affermati nella maggioranza delle NoC trattate come meccanismi utili a regolare il traffico all'interno dell'architettura e i due sistemi presentati (CB,CTC) presentano due facce della stessa medaglia per avere un flow control prestazionale. Mentre il primo ottimizza la banda e la latenza al costo di maggiori risorse, il secondo diminuisce drasticamente i componenti richiesti per l'implementazione, ottimizzando volumi,costi e consumi.

Bibliografia

[1] W.Dally and C.Seitz,"Deadlock-free message routing in multiprocessor interconnection networks",1987

[2] U.Y.Ogras,J.Hu,and R.Marculescu, "Key research problems in NoC design: a holistic perspective", 2005

[3] E.Beign,F.Clermidy,P.Vivet,A.Clouard,and M.Renaudin,"An asynchronous NOC architecture providing low latency",2005

[4] Arteris,"A comparison of network-on-chip and busses",White paper, 2005

[5] A.R.dulescu,J.Dielissen,S.Gonz Pestana, "An efficient on-chip NI offering guaranteed services, shared-memory,abstraction, and flexible network configuration", 2005

[6] Y.H.Song and T.M.Pinkston,"A progressive approach to handling message-dependent deadlock in parallel computer systems", 2003

[7] Andreas Hansson, Kees Goossens,and Andrei Radulescu,"Avoiding Message-Dependent Deadlock in Network-Based Systems on Chip",2006

[8] R.Sivaram, R.Kesavan,"Where to provide support for efficient multicasting in irregular network:NI or switch",1998

[9] Intel,"An Introduction to the Intel QuickPath Interconnect", January 2009

[10] L.Benini,G.De Micheli,"Network on chips:a new SoC paradigm",2002

[11] L.Benini,G.De Micheli,"A new paradigm for System on chips design",2002

[12] A.Agarwal,C.Iskander,R.Shankar,"Survey of Network on Chip (NoC) Architectures and Contributions",2009

[13] B.Bijlsma,"Asynchronous Network-on-Chip Architecture Performance Analysis",2005

[14] T.Bjerregaard,"The MANGO Clockless Network-on-Chip:Concepts and Implementation",2005

[15] T.Bjerregaard,J.Sparso,"A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip",2005

[16] M.Millberg,E.Nilsson,R.Thid,J,Oberg,Z.Lu,A.Jantsch,"The Nostrum Network on Chip",2004

[17] S.Murali,G.De Micheli,"An Application-Specific Design Methodology for STbus Crossbar Generation",2005

[18] V.Zaccaria,"Data-Flow Deadlock Avoidance for Streaming Applications Mapped on Network-on-Chips",2005

[19] S.Warnakulasuriya,M.Pinkston,"A Formal Model of Message Blocking and Deadlock Resolution in Interconnection Networks",2000

[20] S.Murali,P.Meloni,F.Angiolini,D.Atienza,S.Carta,L.Benini,G.De Micheli,L.Raffo,"Designing Message-Dependent Deadlock Free Networks on Chips for Application-Specific Systems on Chips",2006

[21] M.Pinkston,Y.H.Song,"Efficient Handling of Message-Dependent Deadlock",2001

[22] B.Gebremichael,F.Vaandrager,M.Zhang,K.Goossens,"Deadlock Prevention in the AETHEREAL Protocol,2005

[23] R.Boppana,S.Chalasani,C.S.Raghavendra,"Resource Deadlocks and Performance of Wormhole Multicast Routing Algorithms",1998

[24] F.Steenhof,H.Duque,B.Nilsson,K.Goossens,R.Peset,"Networks on chips for high-end consumer-electronics TV system architectures",2006

[25] S.Gonzalez Pestana,E.Rijpkema,A.Radulescu,K.Goossens,O.P.Gangwal,"Cost-performance trade-offs in networks on chip: a simulation-based approach",2004

[26] J.Dielissen,A.Radulescu,K.Goossens,"Power measureaments and analysis of a network-on chip", 2005

[27] STMicroelectronics,"STBus Interconnect",2005

[28] A.Hansson,K.Goossens,A.Radulescu,"A unified approach to constrained mapping and routing on network-on-chip architectures",2005

[29] OCP,"Open Core Protocol",http://www.ocpip.org

[30] A.V.de Mello,L.Copello,F.Moraes,N.L.V.Calazans,"Evaluation of Routing Algorithms on Mesh Based NoCs",2004

[31] "Xpipes,a Network on chip Architecture",http://www.diee.unica.it/eolab2/NoC.html

[32] M.Dall'Osso,G.Biccari,L.Giovannini,D.Bertozzi,L.Benini,"Xpipes:a latency insensitive parametrized Network-on-Chip Architecture for multiprocessor SoC",2003

[33] U.Feige,P.Raghavan,"Exact analysis of Hot-Potato Routing",1992

[34] N.Concer,L.Bononi,M.Soulie,R.Locatelli,"CTC: an End-To-End Flow Control Protocol for Multi-Core Systems-on-Chip",2009

[35] S.Murali,G.De Micheli,,'An Application-Specific Design Methodology for STbus Crossbar Generation',2005

[36] A.S.Tanenbaum,"Computer Networks",1996.

[37] F.P.Trot,A.Greiner,P.Gomez,"On cache coherency and memory consistency issues in NoC based shared memory multiprocessor SoC architectures", 2003

[38] A.S.Tanenbaum,'Architettura dei calcolatori',4th edition, 2002