OCCAM: Fault tolerant link

di Luciano Macera

Se avete letto, come crediamo, il numero di dicembre di MCmicrocomputer, avrete sicuramente notato l'articolo del prode AdP sulla demo fault tolerant, messa a punto dalla INMOS, basata su una pipeline con nodi a tripla ridondanza. In quella demo, oltre al vero e proprio fault di un «intero» transputer, era prevista anche la possibilità di ripristinare il collegamento di link fisici momentaneamente scollegati. Questo mese vedremo un po' più da vicino il problema, svelandovi un po' di trucchetti per realizzare in OCCAM meccanismi di questo tipo

La storia

Già, ma chi non ha letto il numero di dicembre verrà automaticamente tagliato fuori anche dalla lettura di questo articolo? Sicuramente no, e per loro (ma solo per questi) riassumeremo brevemente il funzionamento della demo.

Premesso, come sottolineato dallo stesso AdP, che non si tratta di un prodotto commerciale ma, appunto, di una demo, il dispositivo in questione in pratica serviva per calcolare in tempo reale la FFT su un segnale digitale proveniente da un convertitore A/D collegato ad un comunissimo riproduttore a cassette. Una pipeline a due stati elaborava il segnale mostrando poi a video le frequenze presenti sottoforma di linee verticali colorate che scorrevano sullo schermo. Ogni nodo della pipeline era a sua volta composto da tre moduli d'elaborazione (ognuno dotato di un transputer) che elaboravano i medesimi dati di ingresso col medesimo algoritmo producendo altrettanto identici risultati in uscita. Questo, naturalmente, fintantoché non si verificava alcun malfunzionamento in uno dei sottonodi di un singolo stadio. Nel caso, invece, di fault di uno dei transputer il modulo guasto veniva isolato tramite un meccanismo di voting (se un risultato è diverso dagli altri due, a loro volta identici, è estremamente probabile che il primo provenga da un chip in stato comatoso...) e iniziava la fase di emergenza in cui, sempre senza interrompere il funzionamento

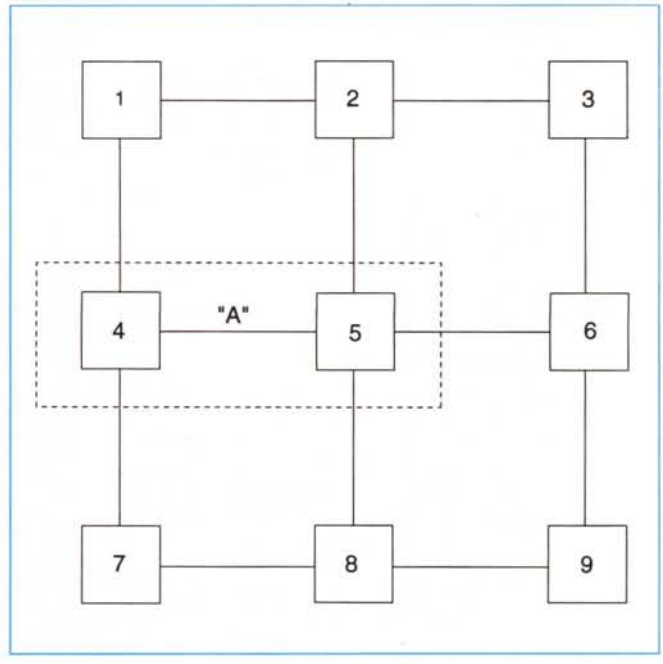
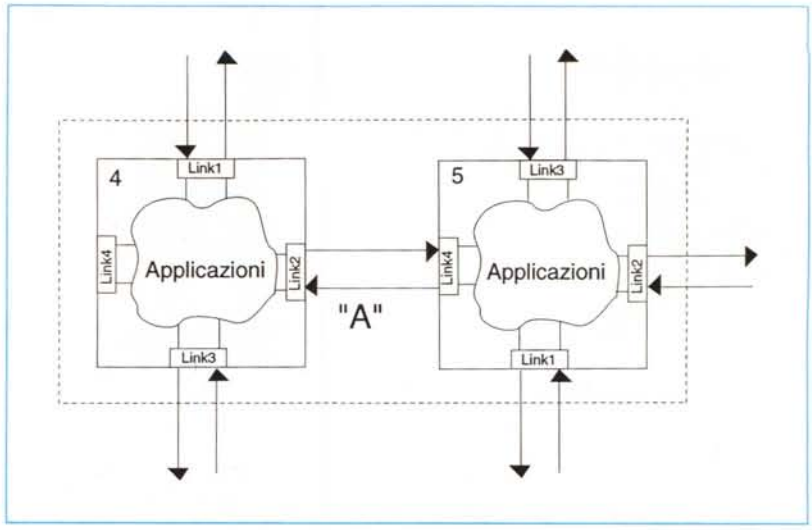


Figura 1
Una generica rete di transputer.

del sistema, i transputer collegati al transputer rotto provavano in continuazione a resettarlo e a farlo ripartire dopo aver spedito ad esso una copia del programma e dei dati aggiornati.

Se l'errore era dovuto, ad esempio, ad una vera e propria rottura di un chip, l'operatore poteva comodamente sostituirlo (togliendo l'alimentazione soltanto a quel modulo) lasciando poi al rimanente sistema il compito di farlo ripartire in sincrono con tutti gli altri transputer.

Come detto all'inizio, oltre al vero e proprio fault di un transputer venivano considerati, e perfettamente assorbiti, anche le temporanee cadute del link fisico tra due transputer. Staccando manualmente il collegamento fisico, il sistema era in grado di accorgersene (distinguendolo quindi dalla rottura di un modulo) e di rieffettuare la sincronizzazione non appena il collegamento fisico fosse stato ripristinato.



▲ *Figura 2 - Particolare «ingrandito» di figura 1. Normalmente i processi accedono direttamente ai link fisici per effettuare comunicazioni inter processor.*

Detto questo

Vediamo come è possibile implementare un meccanismo per tollerare temporanee cadute del supporto fisico di un link. In figura 1 è mostrata una porzione di una generica rete di transputer. Focalizziamo la nostra attenzione, ad esempio, sui transputer 4 e 5 ed in particolare sul link «A» (bidirezionale, come sempre) esistente tra i due.

In un sistema non dotato di meccanismi atti a prevenire cadute di supporto la situazione è grossomodo quella mostrata in figura 2: uno o più processi (lì mostrati genericamente con la nuvoletta «Applicazioni») accedono direttamente ai link fisici utilizzati eseguendo normali operazioni di send e receive (giustamente), come se si trattasse di comunicazioni tra processi in esecuzione su uno stesso transputer. Così un processo del transputer 4 dialoga con un suo partner sul transputer 5 utilizzando il collegamento fisico «A» che, ad esempio, collega il link 2 del primo chip con il link 4 del secondo chip.

Se, però, nel bel mezzo di una comunicazione il collegamento «A» (in figura 2 «A» è sia la freccia diretta a destra che quella diretta a sinistra) viene temporaneamente interrotto, è altamente probabile che i link fisici dei transputer 4 e 5 collegati appunto da «A» vadano in errore rifiutandosi di riprendere a comunicare fino a nuovo reset (dei link). Questo perché vi è una precisa sincronizzazione tra ogni singolo byte inviato in una direzione e corrispondente Ack fisico di risposta dal link corrispondente che ha ricevuto il byte in questione. Quindi la stessa linea «in uscita» (freccia superiore di «A») dal chip 4 al chip 5 è utilizzata

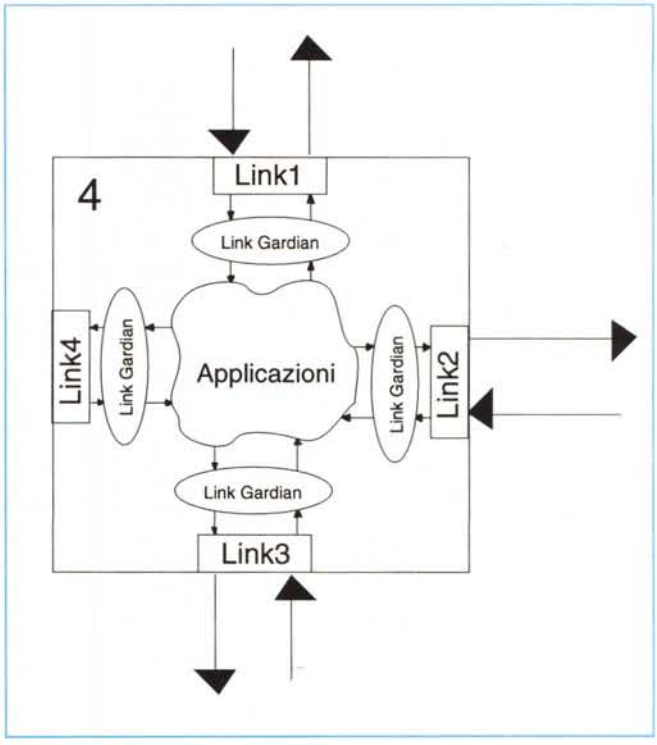


Figura 3 - Interponendo un LinkGardian tra le applicazioni e i link fisici possiamo implementare un meccanismo di autoripristino in seguito a temporanea caduta del supporto fisico. ▶

sia per trasferire messaggi in questo verso che per inviare gli Ack fisici dei messaggi in transito da 5 verso 4 (freccia inferiore). È chiaro che una interruzione temporanea del collegamento «A» può indurre errori di trasmissione non trascurabili, come pezzi di messaggio interpretati come Ack fisici o cose simili. Senza scendere ulteriormente in particolari (semmai ne riparleremo in un articolo futuro) vediamo come è possibile risolvere questo genere di problemi.

Il LinkGardian

Interponendo un opportuno processo «LinkGardian» tra le applicazioni in esecuzione sul transputer ed ogni singolo link fisico (figura 3) è possibile implementare un meccanismo di autoripristino senza nemmeno modificare i processi esistenti e costituenti la già citata «nuvoletta». È addirittura possibile effettuare «l'upgrade» senza nemmeno ricompilare i processi esistenti ma semplicemente compilando e linkando a parte il processo LinkGardian modificando poi solo il file di configurazione processi e canali (descritto lo scorso numero, ricordate?). In pratica i processi esistenti, invece di utilizzare direttamente i link fisici per le loro comunicazioni extra-processor invieranno i dati da spedire al (e riceveranno quelli in arrivo dal)

corrispondente processo LinkGardian che opera su quel particolare link fisico. Non esistendo a livello di processi alcuna differenza tra comunicazioni sullo stesso transputer e tra comunicazioni tra transputer differenti, i processi costituenti la «nuvoletta» Applicazioni non avranno in pratica coscienza dell'irrobustimento dell'intero sistema.

Il che, come sempre, non è poco.

Il funzionamento del processo LinkGardian è abbastanza semplice. In pratica finché tutto funziona a dovere non fa assolutamente nulla. Ciò che riceve (figura 4 A) dal suo canale di ingresso lo inoltra sul link d'uscita, quello che arriva dal link d'ingresso lo inoltra sul suo canale d'uscita.

Il tutto realizzato in maniera parallela (in pratica il processo LinkGardian crea due processi figli, paralleli, «mittente» e «destinatario», detti anche Tx e Rx) fintantoché non si verifica un errore (in pratica un eccessivo ritardo di risposta, l'Ack fisico) sul link.

Se, invece, si verifica un errore dovuto all'assenza di collegamento, i due processi Tx e Rx terminano (il primo per timeout, il secondo avvisato dal primo) e il LinkGardian, come mostrato in figura 4 B, sospendendo qualsiasi attività da e verso le Applicazioni, prende il completo controllo del link fisico in errore tentando di ristabilire la comunicazione sincrona. Ovviamente, come già detto precedentemente, il passaggio tra LinkGardian sdoppiato in Tx e Rx a LinkGardian in stato di Recovery (figg. 4 A e 4 B) avviene pressoché contemporaneamente su entrambi i transputer collegati dal momento che il collegamento interrotto è lo stesso e riguarda tutt'e due i chip.

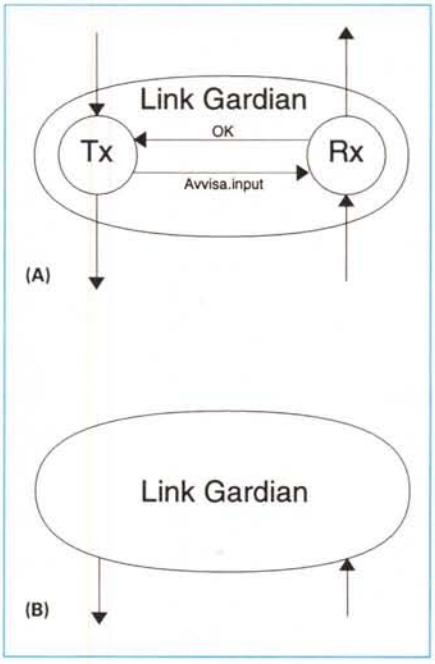


Figura 4 - Il processo LinkGardian si trova nello stato (A) quando il collegamento è «integro», nello stato (B), di Recovery, quando il collegamento fisico è interrotto.

Uno sguardo al listato

Prima di concludere questa breve puntata di Multitasking (è sempre meglio non mettere troppa carne al fuoco benché AdP, dal punto di vista culinario, la pensi ben diversamente...) diamo un'occhiata al listato del processo LinkGardian prima descritto. La numerazione di linea lì presente non è certo un tossicissimo attacco di basic-ite acuta ma è stata aggiunta in fase di stampa del listato per centrare bene le linee che commenteremo.

Ovviamente tutto il funzionamento del LinkGardian è basato sull'utilizzo di funzioni di libreria fornite con il compilatore Occam che permettono di effettuare operazioni non previste direttamente dal linguaggio di programmazione. Queste sono essenzialmente le funzioni Reinitialise(), InputOrFail(), OutputOrFail(), la prima per reinizializzare un link fisico, le rimanenti due per tentare (senza rimanere «appesi») una comunicazione attraverso un link.

Il processo LinkGardian (linee 1 e 2) al suo lancio riceve come parametri i quattro canali che utilizzerà: due mappati su un preciso link fisico, gli altri due assegnati anche ai processi della «nuvoletta» Applicazioni che dovranno spedire e ricevere i loro messaggi.

In questo esempio il protocollo utilizzato per i canali è «array di byte di lunghezza variabile», già illustrato nelle scorse puntate. La dichiarazione dei canali di questo tipo avviene quindi nel seguente modo:

```
CHAN OF INT::[]BYTE nomecanale:
```

Seguono, linee 6..9 alcune dichiarazioni necessarie al funzionamento del processo. Da segnalare (linea 9) i due canali di tipo BOOL utilizzati dai processi figli Tx e Rx (figura 4 A).

Alla linea 11 inizia l'esecuzione del processo che è un loop infinito. Non è prevista, dunque, la possibilità di terminazione per questo processo (a meno di non resettare tutto...). Dopo l'inizializzazione delle due variabili booleane alla linea 14, con il PAR di linea 16 (ulteriormente identificato nel commento dai caratteri «{1}») vengono ufficialmente lanciati in parallelo i due processi Tx e Rx, in pratica il primo alla linea 30 e il secondo alla linea 73. In altre parole ciò che avviene dalla linea 30 in poi avviene contemporaneamente a quello che succede dalla linea 73 in poi (della serie «OCCAM IS MAGIC!!!»). Analizziamo singolarmente i processi Tx e Rx, cominciando dal primo. Abbiamo detto che questi due processi fino a quando non avviene un errore sul link (da qui i WHILE di linea 30 e 73 sulle due variabili booleane «aborted1» e «aborted2» entrambe inizializzate a FALSE) non fanno altro che reinoltrare in uscita ciò che ricevono in ingresso. Quindi alla linea 35 il processo Tx riceve il messaggio da spedire e alla linea 40, utilizzando una funzione di libreria, tenta la comunicazione sul link. La procedura «OutputOrFail.t» ritorna nel suo quinto parametro il valore FALSE se tutto è andato bene, TRUE se entro il tempo indicato nel quarto parametro non è riuscita ad effettuare la spedizione. Segue, sempre nel processo Tx dalla linea 43 in poi, un IF con il quale in caso di errore (aborted1 - TRUE) viene fatto abortire anche il processo Rx (come vedremo), mentre in caso di normale funzionamento (aborted1 - FALSE) si riesegue il loop di linea 30.

E passiamo al processo Rx (linea 73). Il funzionamento è complementare (prima si riceve dal link e poi si invia sul canale se non c'è stato errore), ma l'abort non è provocato dallo scadere di un timeout ma da apposita segnalazione sul canale «Avvisa.Input» da parte del processo Tx (figura 4 A). La funzione di libreria utilizzata è la procedura:

```
InputOrFail.c
```

Utilizzata alla linea 81. A questa passiamo il canale mappato sul link fisico (input.link), un array per ricevere il messaggio in arrivo (message.in), un canale per l'abort (Avvisa.Input) e la solita variabile booleana (aborted2) nella quale troveremo l'esito dell'operazione. Segue, come nel processo Tx, l'IF sulla variabile aborted2 che in caso di TRUE (operazione fallita) restituisce sul canale OK (figura 4 A) un cenno di abort avvenuto, in caso di FALSE, ovvero di operazione effettuata con successo, il messaggio ricevuto dal link viene inoltrato sul canale verso le applicazioni.

Riassumendo, appena si verifica un errore nella «OutputOrFail.t» del processo Tx, questo avvisa (facendogli abortire la «InputOrFail.c») il processo Rx terminando entrambi a causa delle variabili aborted1 e aborted2 tutt'e due a TRUE che non concedono altri «giri» ai due while di linea 30 e 73. Terminati Tx e Rx termina di conseguenza anche il PAR di linea 16 che li aveva lanciati e l'elaborazione continua dalla linea 94 in poi (stato di Recovery, figura 4 B). A questo punto, però, occorre non dimen-

```
1   PROC Link.Gardian(CHAN OF INT::[]BYTE input.link,output.link,
2                                        canale.in,canale.out)
3
4     -- dichiarazione variabili:
5
6     INT                    len:
7     BOOL                   aborted1,aborted2,ok:
8     [MESSAGE.LEN]BYTE      message.in,message.out:
9     CHAN OF BOOL           Avvisa.Input,OK:
10
11    WHILE TRUE  -- loop infinito
12      SEQ
13
14        aborted1,aborted2 := FALSE,FALSE   -- inizializzazione
15
16        PAR  -- {1}
17  .           -- questo PAR lancia i processi "mittente" e "destinatario" e
18            -- termina in caso di errore sul link fisico
19
20
21          --     ------------------------------------------
22          --     ------------------------------------------
23          --
24          --              Processo Mittente (Tx)
25          --      spedizione messaggio al transputer partner
26          --
27          --     ------------------------------------------
28          --     ------------------------------------------
29
30          WHILE (NOT aborted1) -- fintantoche' non si verifica
31            SEQ                -- un errore su "output.link"
32
33              -- ricevo messaggio da spedire:
34
35              canale.out ? len::message.out
36
37              -- provo a spedire il messaggio sul link fisico:
38              -- la funzione fallisce per "timeout".
39
40              OutputOrFail.t(output.link,[message.out FROM 0 FOR len],clock,
41                             timeout,aborted1)
42
43              IF           -- se...
44
45                aborted1   -- ...e' avvenuto un errore su output.link
46
47                  PAR
48                    Avvisa.Input ! TRUE  -- avviso il processo destinatario
49                                         -- (partner nel "PAR" {1} )
50                                         -- facendogli abortire la "receive"
51                    OK ? ok              -- aspetto conferma di "abort"
52
53                  TRUE   -- equivalente ad un "ELSE"
54                    SKIP -- messaggio recapitato, quindi tutto OK!
55
56
57              -- eventuale continuazione del processo mittente...
58              -- ...
59              -- ...
60              -- ...
61
62
63
64          --     ------------------------------------------
65          --     ------------------------------------------

66          --
67          --              Processo Destinatario (Rx)
68          --       ricezione messaggio dal transputer partner
69          --
70          --     ------------------------------------------
71          --     ------------------------------------------
72
73          WHILE (NOT aborted2)     -- fintantoche' non si verifica
74            SEQ                    -- un errore su "input.link"
75
76              -- provo a ricevere un messaggio sul link fisico:
77              -- la funzione fallisce in caso di messaggio sulla
78              -- porta "Avvisa.Input" da parte del processo
79              -- mittente ( partner nel "PAR" {1} ).
80
81              InputOrFail.c(input.link,message.in,Avvisa.Input,aborted2)
82              IF
83                aborted2
84                  OK ! TRUE
85                TRUE         -- messaggio ricevuto, quindi tutto OK!
86                  canale.out ! (SIZE message.in)::message.in
87
88              -- eventuale continuazione del processo destinatario...
89              -- ...
90              -- ...
91              -- ...
92
93
94        -- ------------------------------------------------------------------
95        -- in questo punto del processo "Link.Gardian" ci si arriva in
96        -- caso di errore sul link fisico. E' necessario reinizializzarlo
97        -- ed effettuare la risincronizzazione dei corrispondenti processi
98        -- in esecuzione sui due transputer fisicamente collegati dal link.
99        -- Reinizializzazione e risincronizzazione effettuata simultaneamente
100       -- su entrambi i transputer, in quanto un errore di input per un
101       -- transputer provoca un errore di output sul secondo transputer
102       -- ------------------------------------------------------------------
103
104       WHILE aborted1 OR aborted2  -- {2}
105         SEQ                       -- finantoche' persiste l'errore...
106           PAR
107             Reinitialise(input.link)  -- reinizializzazione in parallelo
108             Reinitialise(output.link) -- dei link fisici in stato d'errore
109
110         delay := ...  -- costante di tempo da definire opportunamente
111
112         clock ? AFTER delay          -- attesa
113
114         -- prova di sincronizzazione:
115
116         PAR
117           InputOrFail.t(input.link,temp,clock,timeout,aborted1)
118           OutputOrFail.t(output.link,"Ehi, tu, mi senti ?",
119                          clock,timeout,aborted2)
120
121
122         -- ------------------------------------------------------
123         -- si esce dal loop {2} quando il collegamento fisico
124         -- e' stato ripristinato e i due transputer sono riusciti a
125         -- risincronizzarsi dopo aver resettato i link
126         -- ------------------------------------------------------
127
128
129       -- si ritorna al loop infinito di cui sopra...
```

Listato del processo LinkGardian.

ticare che quanto successo sul transputer in questione avverrà più o meno simultaneamente sul transputer partner: se manca il collegamento fisico tra i due chip, su entrambi si verificherà l'errore sui link e quindi lo stato di Recovery dei corrispondenti LinkGardian.

Alla linea 104 troviamo il loop principale di questo stato che continua a persistere fintantoché le variabili aborted1 e aborted2 non sono tutt'e due FALSE. La prima operazione da compiere (linee 106..108) è quella di reinizializzare in parallelo (tenete sempre a mente che tutto ciò sta avvenendo anche sull'altro transputer) tanto l'intput.link quanto l'output.link. Atteso un opportuno intervallo di tempo (linee 110..112) sempre in parallelo si tenta una comunicazione con il transputer partner che a sua volta starà tentando una comunicazione con noi. Da notare che in questo caso tutt'e due le operazioni di XxxxxOrFail sono di tipo «.t» quindi abortiscono in caso di timeout scaduto. Sarebbe opportuno (da questo la linea 110 incompleta) avere un delay leggermente diverso per ogni chip in modo da scongiurare, o quantomeno ridurre, fenomeni di «rimbalzo perpetuo» in cui ogni volta che un chip resetta il link l'altro tenta la comunicazione e viceversa, non riuscendo mai a risincronizzarsi.

Bene, anche per questo mese mettiamo qui la nostra parola chiave «FINE» dandovi appuntamento ancora ai prossimi numeri per immergerci sempre di più in questo fantastico mondo (se siete arrivati fin qui la penserete come me, AdP e... tant'altri) della programmazione parallela. MC

Specialisti in duplicazione

La Microforum di Toronto, Canada, produttrice dei famosi dischetti Mito, propone oggi al mercato italiano del software i suoi sofisticati impianti di duplicazione. Nel giro di pochi giorni, Microforum può assicurare la duplicazione dei vostri programmi, anche con protezione, con la massima accuratezza e a costi altamente competitivi. Se il vostro problema sono 1000 o 100.000 copie, scrivete o mandate un fax a

Microforum
MANUFACTURING INC.
TORONTO - CANADA

1 Woodborough Avenue, Toronto, Canada M6M 5A1
Tel. 001 416 656 6406 Fax 001 416 656 6368 Telex (06)23303