Le infrastrutture di rete al giorno d’oggi richiedono l’utilizzo di funzioni complesse e diverse tra loro. Come ad esempio Firewall, Load Balancer o semplici switch.

Un progetto portato avanti dal Network Programmability Laboratory, FlowBlaze, si propone come soluzione per l’implementazione di un data plane programmabile in hardware. I punti di forza di una tale soluzione sono flessibilità in quanto è programmabile e riconfigurabile e velocità avendo rimosso operazioni nel lento layer software. La implementazione in NetFPGA, una smart-nic della digilent, raggiunge un processamento dei pacchetti a 40Gbps. Al momento, inoltre, si stà lavorando per importare il lavoro svolto su schede Intel Altera con frequenze di clock fino ai 600MHz e processamento di pacchetti teoricamente oltre i 100Gbps. (?)

Vediamo ora lo schema di principio dell’architettura di FlowBlaze. Senza entrare troppo nel dettaglio, prendiamo in considerazione solo la Flow-Context Table, la tabella nella quale vengono salvati lo stato e il contesto di ogni flusso che attraversa la scheda. In particolar modo essa è stata implementata attraverso una cuckoo hash table. L’utilità dell’algoritmo di cuckoo è quello di poter avere 1) tempo di inserimento e ricerca costante e pari ad un colpo di clock e 2) una space utilization pari a circa il 97% della tabella.

Una grande limitazione di questa hash table è la staticità delle dimensioni. Nonostante infatti FlowBlaze sia riconfigurabile in modo da implementare qualunque tipo di funzione di rete stateful, non abbiamo possibilità di definire le dimensioni delle chiavi da salvare nella Flow Context Table. Il mio lavoro di tesi è stato quello di ottimizzare tale hash table ideando un modulo che ne rendesse dinamiche le dimensioni.

Una tabella dinamica dà maggior flessibilità di utilizzo potendone configurare le dimensioni per ogni funzioni da implementare.

Avremo una riconfigurazione incredibilmente veloce dovendo semplicemente variare il valore di un segnale durante l’uso al posto di istanziare memorie diverse e dover passare nuovamente attraverso il processo di sintesi

Aumenteremo la space efficiency massimizzando l’utilizzo di celle di memoria allocate in fase di sintesi

Ogni elemento stateful in una pipeline di FlowBlaze avrà la possibilità di definire i propri valori di dimensione di chiave e valore.

A tale scopo inizialmente ho studiato l’architettura FlowBlaze e la cuckoo hash table usata. Ho poi progettato il mio modulo VHDL che la rendesse dinamica e riconfigurabile a run-time. Ho simulato e sintetizzato prima il mio modulo e poi l’ho integrato nel prototipo di FlowBlaze simulando e sintetizzando il progetto nuovamente. Infine ho eseguito un caso d’uso caricando il progetto su una NetFPGA ed eseguendo un ping tra due interfacce di rete e FlowBlaze integrato della hash table riconfigurabile. Il test è stato eseguito provando 5 possibili casi di dimensione diverse di key e value.

L’astrazione alla base di ciò che si è voluto realizzare è mostrato nella seguente slide. Una normale RAM ha un certo numero di locazioni indirizzabili da un address\_bus. In questo esempio le parole che stiamo salvando occupano metà dello spazio allocato in memoria per ognuna di esse. Vogliamo idealmente poter “tagliare” a metà la RAM e allungarla andando difatto a raddoppiare il numero di parole totali che potremo salvare.

Uno schema a blocchi di come tale astrazione sia stata realizzata in hardware è il seguente.

#Entra più nel dettaglio di quello che hai fatto.

Nella hash table in ogni entry viene salvata la coppia key e value; FlowBlaze usa RAM dual port in modo da poter inserire e cercare contemporaneamente chiavi all’interno della tabella. Staticamente si era deciso di allocare parole a 256 bits usandone 128 per la chiave e 128 per il contesto.

Per raggiungere lo scopo posto ho aggiunto due segnali key\_len e value\_len attraverso i quali dinamicamente viene definito il valore in binario di lunghezza di entrambi. Ho progettato poi un modulo che riconfigurasse le parole e gli indirizzi in ingresso e in uscita alle RAM.

Con un primo case combinatorio sui due segnali introdotti sono riuscito a selezionare solo i bit utili dei vettori key e value in ingresso. Con un secondo case combinatorio su alcuni bit dell’indirizzo in ingresso inoltre ho potuto ripartire la parola in ingresso tra le due o più porzioni di memoria individuate.

Per poter scrivere le nuove parole senza intaccare quelle già presenti inoltre è stato utilizzato il write enable a byte con il quale si possono selezionare byte a byte quali porzioni della memoria da sovrascrivere.

Nell’esempio riportato, in ogni riga della Ram fisica saranno salvate effettivamente due parole. Con il bit meno significativo dell’indirizzo si definiranno le due “zone” di destra e di sinistra della memoria. Un address in ingresso di valore 3 dunque punterà alla prima riga della Ram e alla parte di sinistra.

Dopo una prima fase di progettazione ho effettuato alcune simulazioni. Per testare il corretto funzionamento di una RAM si alternano una fase di scrittura e una di lettura di tutta la memoria. In scrittura ad indirizzo n viene scritto in binario il valore associato ad n in modo tale da poter facilmente capire in lettura se funziona o meno. Come vediamo al colpo di clock evidenziato all’indirizzo 30 scriviamo una word con entrambi key e value con valore 30. In lettura scorriamo tutta la memoria e rileggiamo l’uscita dalle due porte della RAM per testare il funzionamento di entrambe. A scorre ad indirizzi crescenti mentre B decrescenti e ad indirizzo dato corrisponde esattamente l’uscita della porta assegnata.

Testiamo quindi il funzionamento a dimensione diversa cambiando prima il valore associato a key\_len e value\_len e successivamente ripetendo il processo di scrittura di tutta la memoria e nuovamente lettura da entrambe le porte. Difatti notiamo che funziona a tutte le dimensioni quindi effettivamente allochiamo più locazioni di memoria di quante non ne abbia la RAM vera e propria.

Dopo la simulazione si è passato all’implementazione e alla sintesi sulla FPGA. Vediamo un aumento di risorse usate da hash table statica e dinamica. E infine vediamo un confronto in utilizzo dell’area di FlowBlaze e la hash table dinamica. Sono riportate numero di LUTs usate e potenza usata da entrambi.

Per concludere si è testato il funzionamento della Hash Table dinamica con un caso d’uso di FlowBlaze usato come semplice dispositivo di switch. In particolare, il test si è basato sull’invio e la ricezione di un ping tra due interfacce di rete. Come primo passo sono stati associati alle interfacce dei MAC e IP addresses.

Si è poi passato a caricare il bitstream di FlowBlaze sulla NetFPGA attraverso lo script sh program\_fpga.sh.

Flowblaze è stato programmato per salvare nella Flow Context Table come chiave l’indirizzo MAC di destinazione e come contesto la porta d’uscita sulla quale è presente il device cercato. Per fare ciò sono stati scritti, attraverso protocollo AXI, i registri della scheda. In particolare è stato aggiornato il registro, conoscendone l’indirizzo, associato ai segnali key\_len e value\_len. Nello specifico 0x2060 corrisponde ad un valore di 32 bits (20 in hex) per il contesto e 96(60 in hex) per la chiave. Infine è stato inviato il segnale di ping da eth2 verso l’IP address di eth3.

Avendo il ping dato esito positivo, la hash table ha correttamente salvato i MAC address e ha associato ad esse le porte della scheda su cui tale dispositvo è effettivamente presente. Vediamo adesso il tcpdump di entrambe le interfacce di rete da cui possiamo individuare un corretto invio di pacchetti tra di essi.

In conclusione è stato presentato il problema di staticità associato alle dimensioni delle Hash Tables di FlowBlaze

È stata descritta l’astrazione di una HT riconfigurabile dinamicamente

Sono state eseguite simulazioni di 5 diverse configurazioni della tabella

È stato integrato il modulo per la riconfigurazione dinamica in FlowBlaze

Infine, è stato eseguito un caso di utilizzo della riconfigurazione su NetFPGA

Come sviluppi futuri sono stati prefissati di importare il progetto su Intel FPGA

Utilizzo di memorie esterne per salvare un maggior numero di entries

#Tramite bus AXI andremo ad aggiornare i valori salvati nel registro a 32 bit in modo da cambiare dimensione alla Hash table. Conoscendo da progetto l’indirizzo di tale registro, ciò può essere fatto dal semplice script sh rwaxi

F

# Ad uno stato e un ingresso associamo uno stato successivo e un’uscita.

# Le funzioni di rete vengono definite attraverso una leggera rivisitazione di una classica macchina a stati, tipo mealy machine.

#Nella hash table viene salvata la coppia key e value ad ogni indirizzo. Di ogni pacchetto in ingresso viene controllato un campo che lo contraddistingue da pacchetti di altri o dello stesso flusso. Questa porzione di pacchetto è la chiave. Ad ogni chiave è associato un value nel quale saranno salvati stato e contesto di tale flusso. Attraverso i due segnali key\_len e value­\_len specifichiamo le dimensioni che avranno la chiave e il valore e di conseguenza la dimensione della memoria. FlowBlaze utilizza RAM dual-port in modo da poter eseguire contemporaneamente operazioni di insert e search nella hash table (questa contemporaneità è necessaria soprattutto per avere un cuckoo con lo stash funzionante). Avremo quindi i due segnali key e search\_key per salvare nuovi flussi o cercarne di già presenti. Attraverso un modulo combinatorio di cui non entreremo nel dettaglio associamo ad ogni chiave in ingresso un indice della memoria attraverso una hash function.

# Con questo tipo di astrazione, proposta come OpenFlow, si rendeva possibile l’utilizzo di hardware di vendors diversi per implementare qualunque tipo di funzione di rete. Tuttavia sono state compromesse le performance dovendo eseguire molte operazioni nel lento livello software L’astrazione iniziale di OpenFlow, con la quale si è proposta la softwareizzazione della gestione delle funzioni, può essere rivisitata ampliando il semplice concetto delle tabelle match action e implementando le XFSM

#Vediamo nel dettaglio l’implementazione hardware di flowblaze composta da una Flow context table nel quale verrà salvato lo stato e il contesto associato ad ogni flusso e una XFSM table nella quale per ogni stato, valore del contesto ed evento associa un next state, azione e update del valore del contesto.

# Una volta testato il funzionamento, la hash table è stata integrata in FlowBlaze. Ne è stato testato il funzionamento da hash table, ovvero se cercando una chiave precedentemente salvata, asserisce il bit di match e manda in uscita il contesto legato a tale chiave.

#; inizialmente esse erano realizzate da devices con hardware dedicato per ognuna di queste funzioni diverse (firewall, router, load balancer…). Avremmo avuto un device specifico che operava come switch e solamente come switch. Si nota come questa sia una soluzione costosa e poco flessibile

In un secondo momento, per incrementare la flessibilità dell’hardware e quindi abbassarne i costi, si è proposta una softwarizzazione della gestione delle funzioni di rete. Utilizzando devices general purpose (quindi molto più economici) venivano definite le funzioni da un livello di controllo logicamente centralizzato e gestito in software. Definendo macroregole di funzionamento, microregole venivano iniettate nei singoli dispositivi. La softwarizzazione è un’ottima soluzione per semplificare la gestione e avere costi ridotti da devices general purpose, TUTTAVIA è lenta (fattore di 50 tra velocità software e hardware).

#. . Essa è stata inizialmente pensata come tabella statica con dimensioni fisse.

# Il mio lavoro di tesi si è basato sull’ottimizzazione della Hash Table implementando un modulo per la riconfigurazione dinamica delle dimensioni della tabella, in base alle varie tipologie di entries da salvare. L’astrazione di ciò che è stato realizzato è presentata in questa slide. Dinamicamente possiamo cambiare le dimensioni delle parole da salvare e la memoria verrà cambiata di conseguenza incrementando il numero di entries totali salvabili e migliorando la space utilization.

# Di ogni pacchetto in ingresso viene controllato un campo che lo contraddistingue da pacchetti di altri o dello stesso flusso. Questa porzione di pacchetto è la chiave. Ad ogni chiave è associato un value nel quale saranno salvati stato e contesto di tale flusso.

# Avremo quindi i due segnali key e search\_key per salvare nuovi flussi o cercarne di già presenti. Attraverso un modulo combinatorio di cui non entreremo nel dettaglio associamo ad ogni chiave in ingresso un indice della memoria attraverso una hash function. L’address\_a è usato per cercare keys mentre address\_b è usato per scrivere