OPTIMALIZACE VYHLEDÁNÍ NEJDELŠÍHO PREFIXU SÍŤOVÉ ADRESY S VYUŽITÍM ČÁSTEČNÉ DYNAMICKÉ REKONFIGURACE FPGA

Jiří Matoušek
Výpočetní technika a informatika, 1. ročník, prezenční studium
Školitel: Zdeněk Kotásek

Fakulta informačních technologií, Vysoké učení technické v Brně
Božetěchova 1/2, 612 66 Brno

imatousek@fit.vutbr.cz

Abstrakt. Článek se zabývá optimalizací rychlosti operace vyhledání nejdelšího shodného prefixu pomocí algoritmu Tree Bitmap. Optimalizace jsou navrhovány tak, aby v maximální možné míře využívaly prostředků současných FPGA čipů. Kromě nahrazení externí paměti pomocí Block RAM paměti umístěné přímo v FPGA je navrženo také rozdělení jednotlivých kroků algoritmu do samostatných stupňů zřetězené linky. Pro výsledné řešení je uveden návrh hardwarové architektury a jsou nastíněny vlastnosti takto optimalizovaného algoritmu. V článku je také obsažena kapitola o směřování mojí disetační práce.

Klíčová slova. LPM, Tree Bitmap, FPGA

1 Úvod

Počet zařízení připojených k Internetu narůstá každým dnem a během několika málo let se očekává úplné vyčerpání adresového prostoru protokolu IPv4 (*Internet Protocol version 4*) [1]. S počtem přidělených IPv4 adres narůstá také počet záznamů ve směrovacích tabulkách routerů. Navíc dochází ke zrychlování komunikace a například pro technologii Ethernet jsou již standardizovány přenosové rychlosti 40 Gb/s a 100 Gb/s [2]. Oba tyto trendy se nepříznivě projevují v požadavcích na vlastnosti routerů především v páteřních sítích. Nejnáročnější operací v routeru je přitom vyhledání nejdelšího shodného prefixu (*Longest Prefix Match*, LPM) dané IP adresy a záznamů ve směrovací tabulce. V páteřních routerech je nutné nasadit k provádění této operace specializované hardwarové řešení, nejčastěji v podobě implementace algoritmu Tree Bitmap [3]. Ačkoliv je tento algoritmus optimalizován tak, aby během jednoho výpočetního kroku přistupoval do paměti pouze jedenkrát, pro výpočet LPM v databázi s dlouhými prefixy je typicky potřeba provést několik výpočetních kroků algoritmu. A právě přístup do externí paměti je při nasazení hardwarového řešení časově a energeticky nejnáročnější.

Hardwarovou implementaci algoritmu lze kromě ASIC (*Application Specific Integrated Circuit*) čipu provést také s využitím technologie FPGA (*Field Programmable Gate Array*). Výhodou FPGA čipů jsou integrované paměťové bloky, které lze využít jako interní paměť a dosáhnout tak při implementaci algoritmu Tree Bitmap úplné eliminace externí paměti. Algoritmus lze dále urychlit zavedením zřetězeného zpracování, které je umožněno díky distribuované povaze paměti na čipu FPGA. Při statickém rozdělení paměti mezi jednotlivé stupně zřetězené linky by však pro pokrytí nejhoršího možného

případu docházelo ke značnému plýtvání omezenými paměťovými zdroji, a proto je vhodné zajistit dynamické přidělování paměti pomocí částečné dynamické rekonfigurace FPGA čipu.

2 Algoritmy LPM

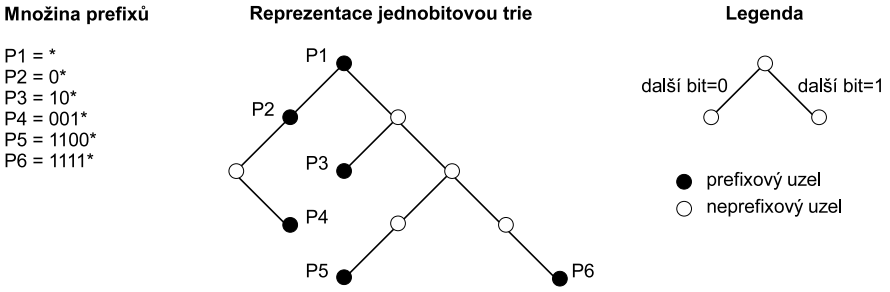
Vstupem LPM algoritmu je daná hodnota a sada prefixů, typicky různě dlouhých. Výstupem algoritmu je pak nejdelší z prefixů obsažený ve specifikované hodnotě. Nejtypičtější využití operace LPM v oblasti počítačových sítí představuje proces směrování prováděný na routerech. V tomto případě je vstupní hodnotou cílová IP adresa paketu a sada prefixů je uspořádána do tzv. směrovací tabulky, ve které jednotlivé záznamy obsahují kromě prefixů IP adres také informace pro další směrování paketu k cílovému zařízení.

Pro vzájemné porovnání algoritmů implementujících operaci LPM budeme obdobně jako v [3] používat tři základní ukazatele: rychlost vyhledání nejdelšího shodného prefixu, paměťové nároky vyhledávacích struktur a rychlost aktualizace vyhledávacích struktur.

2.1 Algoritmus využívající „jednobitovou trie“

Tento algoritmus lze považovat za základ všech dalších uvedených algoritmů. Jejich společným znakem je využití stromové datové struktury nazývané v kontextu LPM algoritmů *trie* (někdy též prefixový strom), která slouží k vyhledávání nejdelšího shodného prefixu.

Kromě zakódování prohledávané množiny prefixů přímo do struktury trie lze na obrázku 1 pozorovat také korespondenci této datové struktury s binárním stromem, přičemž jednotlivé bity vstupní hodnoty určují, zda bude průchod stromem pokračovat v levém nebo pravém podstromu aktuálního uzlu. Binární kód vstupní hodnoty tedy udává cestu stromem a poslední prefix na této cestě ve směru od kořene k listům je hledaným nejdelším shodným prefixem.



Obrázek 1: Reprezentace ukázkové množiny prefixů „jednobitovou trie“

Implementace operace LPM využívající „jednobitovou trie“ vynikají především rychlostí aktualizací. Při použití kompresních metod je i paměťová náročnost algoritmů nízká. Zásadním problémem tohoto přístupu je však rychlost vyhledávání, protože v nejhorším případě odpovídá počet kroků algoritmu počtu bitů vstupního slova.

2.2 Algoritmus Lulea

Základním rozdílem algoritmu Lulea [4] oproti přístupu s „jednobitovou trie“ je zpracování několika bitů vstupního slova v jediném kroku. Označme počet bitů zpracovávaných v jediném kroku jako SL (z anglického termínu *stride length*). Vyhledávací datová struktura je poté tvořena 2^{SL}-arním stromem, jehož uzly obsahují kromě informací o sadě prefixů také informace usnadňující vyhledání nejdelšího shodného prefixu v této datové struktuře.

Reprezentace uzlu stromu v algoritmu Lulea je navržena tak, aby její paměťové nároky byly co nejmenší. Za tímto účelem je využita tzv. technika „*leaf pushing*“, při které se v uzlu stromu zachovává namísto oddělených informací o prefixech a ukazatelích na další uzly stromu jen výběr z těchto informací (pro každou kombinaci hodnot zpracovávaných bitů buď jen ukazatel nebo informace o prefixu), přičemž zbývající hodnoty jsou ve stromu odsouvány na volné pozice směrem k jeho listům. Při využití této techniky lze snížit paměťové nároky algoritmu přibližně na polovinu.

Algoritmus Lulea umožňuje díky zpracování více bitů vstupního slova najednou rychlejší vyhledání nejdelšího shodného prefixu. Jak již bylo naznačeno výše, struktura uzlu vyhledávací datové struktury je pro tento algoritmus optimalizovaná na co nejmenší spotřebu paměti, takže i v tomto ukazateli vykazuje Lulea dobré výsledky. Problémovým parametrem je však rychlost aktualizace stromu, při které může být v nejhorším případě ovlivněna až polovina všech jeho uzlů, což způsobuje technika „leaf pushing“.

2.3 Algoritmus Tree Bitmap

Cílem návrhu algoritmu Tree Bitmap [3] bylo zachovat co nejvíce z pozitivních vlastností Lulea algoritmu a zároveň vylepšit algoritmus z pohledu času potřebného k provedení aktualizace vyhledávací stromové datové struktury. Namísto techniky „leaf pushing“ se tak v uzlech stromu algoritmu Tree Bitmap objevuje další bitmapa, která společně s jedním odkazem na začátek pole synovských uzlů umožňuje uchovávat v uzlu stromu jak informace o prefixech, tak informace o odkazech na potomky.

Ačkoliv je celková paměťová náročnost tohoto algoritmu v porovnání s algoritmem Lulea poněkud horší, uzly stromu mají u Tree Bitmap menší velikost a tato velikost je konstantní. Díky tomu je možné načíst celý uzel z externí paměti v jediném kroku a i pro stejnou hodnotu SL tak dosahovat rychlejšího vyhledání nejdelšího shodného prefixu, než u algoritmu Lulea.

Algoritmus Tree Bitmap tedy dosahuje oproti algoritmu Lulea výrazného vylepšení času potřebného pro aktualizaci trie a díky zmenšení počtu přístupů do paměti také urychlení vyhledání nejdelšího společného prefixu. Cenou za tato vylepšení je vyšší paměťová složitost algoritmu. Celkově však dosahuje algoritmus Tree Bitmap vyváženého výkonu ve všech třech sledovaných ukazatelích a je v současnosti jedním z nejpoužívanějších.

3 Optimalizace LPM algoritmu Tree Bitmap

Počet přístupů algoritmu Tree Bitmap do paměti je v nejhorším případě dán vztahem

$$\left\lceil \frac{\text{délka vstupního slova}}{SL} \right\rceil + 1 \tag{1}$$

V článku [3], kde je algoritmus Tree Bitmap představen, navíc autoři zmiňují, že sami nenastavují parametr SL na hodnotu větší než 8 a pro 32bitovou IPv4 adresu tedy může být potřeba přistoupit do paměti až pětkrát. Nahrazením externí paměti pomocí paměti na čipu FPGA lze tudíž dosáhnout jednak dalšího urychlení operace LPM a navíc také energetické úspory.

Zda je vůbec možné přesunout veškeré vyhledávací datové struktury algoritmu Tree Bitmap z externí paměti do paměti na čipu FPGA ukazují tabulky 1 a 2. V první z nich jsou uvedeny hodnoty dostupné paměti na některých rodinách FPGA čipů firmy Xilinx, jak je uvádí výrobce v [7] a [8]. Kromě celkové dostupné paměti obsahuje tabulka také informaci o tom, do kolika Block RAM bloků je dostupná paměť rozdělena (každý blok obsahuje 36 Kb paměti). Rozsah hodnot na jednotlivých řádcích pak udává rozsah příslušných parametrů pro různé varianty čipů dané rodiny.

Hlavní náplní druhé tabulky jsou paměťové nároky na reprezentaci dvou reálně používaných prefixových sad datovými strukturami algoritmu Tree Bitmap při čtyřech různých hodnotách parametru SL. Tyto hodnoty byly pro potřeby tohoto článku změřeny pomocí nástroje Netbench [6] a u každé prefixové sady je uveden také údaj o její velikosti. Jak je z hodnot v tabulkách 1 a 2 patrné, při vhodné volbě

Tabulka 1: Přehled dostupné Block RAM paměti v různých rodinách FPGA firmy Xilinx (viz [7, 8])

Rodina FPGA	**Dostupná paměť [Kb]**	**Block RAM bloků**
Virtex-5 LXT	936-11 664	26-324
Virtex-5 LX	1 152-10 368	32-288
Virtex-5 FXT	2 448-16 416	68-456
Virtex-5 SXT	3 024-18 576	84-516
Virtex-5 TXT	8 208-11 664	228-324
Virtex-6 LXT	5 616-25 920	156-720
Virtex-6 HXT	18 144-32 832	504-912
Virtex-6 SXT	25 344-38 304	704-1 064

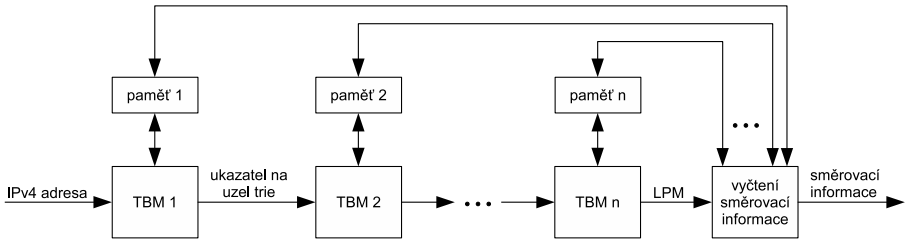
Tabulka 2: Paměťové nároky reprezentace prefixových sad v algoritmu Tree Bitmap

Prefixová sada	**Počet prefixů**	**Paměťová náročnost [Kb]**			
		***SL*=3**	***SL*=4**	***SL*=5**	***SL*=8**
IPv4-space[1]	231 712	9 047,470	10 318,393	5 933,388	56 648,033
AS2.0[2]	416 703	33 276,393	38 330,674	32 042,546	197 934,766

parametru SL je možné v interní paměti FPGA reprezentovat i prefixovou sadu obsahující více než 416 tisíc položek.

Kromě přesunu vyhledávacích datových struktur do interní paměti na čipu FPGA je možné rychlost vyhledání nejdelšího shodného prefixu zvýšit ještě zavedením zřetězeného zpracování, kdy je každý krok algoritmu Tree Bitmap, včetně závěrečného přístupu do paměti pro směrovací informaci příslušející nejdelšímu shodnému prefixu, prováděn samostatnou výpočetní jednotkou v odděleném stupni zřetězené linky. Toto rozšíření je umožněno distribuovanou povahou interní paměti FPGA čipu, takže každému z výpočetních stupňů zřetězené linky může být přidělena samostatná paměť. Přístup do paměťového bloku je pak povolen jen odpovídajícímu výpočetnímu stupni algoritmu Tree Bitmap a jednotce v posledním stupni linky, která zajišťuje vyčtení směrovací informace příslušející nejdelšímu shodnému prefixu. Tento dvojitý přístup lze zajistit díky dvěma vstupně-výstupním portům každého z Block RAM bloků.

Schéma na obrázku 2 znázorňuje hardwarovou realizaci algoritmu Tree Bitmap s aplikovanými výše popsanými vylepšeními. Bloky „TBM i“ a „paměť i“ pro $i \in 1, 2, ..., n$ společně tvoří vždy jeden stupeň zřetězené linky a poslední stupeň obsahuje pouze blok „vyčtení směrovací informace“.



Obrázek 2: Schéma hardwarové realizace optimalizovaného Tree Bitmap algoritmu

Součástí schématu na obrázku 2 není řadič částečné dynamické rekonfigurace a konfigurační rozhraní ICAP (*Internal Configuration Access Port*), ačkoliv se s jejich využitím počítá. Úkolem těchto jednotek je zajistit s využitím částečné dynamické rekonfigurace průběžnou adaptaci kapacity paměťových bloků přiřazených jednotlivým stupňum zřetězené linky podle aktuálních potřeb algoritmu. Pokud bychom totiž určovali velikost jednotlivých paměťových bloků staticky, docházelo by k plýtvání s již tak omezeným množstvím paměti na čipu FPGA. Paměťovou rezervu bychom totiž museli zajistit pro každý blok zvlášť, namísto společné paměťové rezervy v případě využití jedné paměti pro všechny části algoritmu.

3.1 Vlastnosti optimalizovaného algoritmu

Popsané optimalizace hardwarové implementace algoritmu Tree Bitmap s využitím technologie FPGA byly navrženy s cílem urychlit operaci LPM. Optimalizovaný algoritmus je teoreticky schopný dosáhnout v každém taktu vyhledání nejdelšího shodného prefixu pro jiné vstupní slovo. Paměťová náročnost původního algoritmu zůstala zachována, přičemž k zabránění plýtvání s pamětí rozdělenou do jednotlivých stupňů zřetězené linky je využita částečná dynamická rekonfigurace FPGA.

Navržené optimalizace rychlosti operace LPM se pozitivně projevují také v rychlosti provádění aktualizací vyhledávacích datových struktur. Ačkoliv může být aktualizace zdržena provedením částečné dynamické rekonfigurace, eliminuje se toto zdržení výrazným zrychlením přístupu k aktualizované paměti. Díky zřetězenému zpracování navíc může probíhat aktualizace datových struktur paralelně s vyhledáváním nejdelšího shodného prefixu.

3.2 Využitelnost optimalizovaného algoritmu

Jelikož prozatím nebyla provedena hardwarová implementace navržených optimalizací Tree Bitmap algoritmu, je třeba ještě vyčkat na experimentální ověření vlastností popsaných v předchozí podkapitole. Pokud by se však popisované vlastnosti potvrdily, mohla by optimalizovaná hardwarová implementace algoritmu Tree Bitmap nalézt díky podporované rychlosti provádění operace LPM uplatnění v páteřních směrovačích zajišťujících směrování při přenosových rychlostech v řádu desítek až stovek Gb/s.

Nevýhodou řešení je omezené množství paměti dostupné na čipu FPGA, které navíc nelze pro konkrétní čip nijak navyšovat. V nejnovějších FPGA firmy Xilinx (viz [9]) je však opět množství dostupné interní paměti větší, než u rodin popsaných v tabulce 1.

4 Směřování disertační práce

Tématem mojí disertační práce je využití rekonfigurovatelných obvodů v oblasti počítačových sítí. V rámci tohoto tématu bych se chtěl zabývat především použitím částečné dynamické rekonfigurace FPGA pro adaptivní akceleraci výpočetně náročných operací z oblasti počítačových sítí (LPM, vyhledávání vzorů, klasifikace paketů, analýza hlaviček paketů) v závislosti na zatížení výpočetní jednotky (typicky obecný procesor) provádějící danou operaci. V rámci disertace tedy bude nutné řešit především úlohy z oblasti plánování a mapování síťových operací do FPGA.

Postupy navrhované v rámci disertace by měly být prakticky ověřovány při vývoji API (*Application Programming Interface*), které by uživateli umožňovalo starat se pouze o softwarovou část síťové aplikace, přičemž vybrané operace by byly v případě potřeby akcelerovány v rekonfigurovatelném hardware a tato akcelerace by probíhala plně v režii API, transparentně vůči uživateli. Cílovou platformou pro uvažované API tedy musí být systém s obecným procesorem a rekonfigurovatelnou logikou, přičemž aktuálně se jako nejvhodnější jeví platforma Xilinx Zynq-7000 EPP [5].

[1] získáno z `http://bgp.potaroo.net/ipv4-stats/prefixes.txt`
[2] získáno z `http://bgp.potaroo.net/as2.0/bgptable.txt`

5 Závěr

Tento článek se zabývá urychlením operace vyhledání nejdelšího shodného prefixu, přičemž je vycházeno z algoritmu Tree Bitmap a k jeho urychlení jsou využívány prostředky poskytované současnými FPGA čipy. První navrženou optimalizací algoritmu je eliminace externí paměti a její nahrazení pamětí typu Block RAM, která je přímo součástí FPGA čipu. Vedlejším efektem eliminace externí paměti je i energetická úspora. Druhá optimalizace spočívá v rozdělení jednotlivých kroků algoritmu do samostatných stupňů zřetězené linky. Taková optimalizace přitom vyžaduje vlastní paměťový blok pro každý krok výpočtu. Tento požadavek je vyřešen díky distribuované povaze Block RAM paměti.

Myšlenky vztahující se k optimalizaci algoritmu Tree Bitmap jsou v článku doplněny také návrhem hardwarové architektury takto optimalizovaného algoritmu a jsou popsány předpokládané vlastnosti výsledného řešení. Jelikož však navržená architektura nebyla prozatím implementována, nelze tyto předpoklady podložit experimentálními výsledky. Implementování optimalizované verze algoritmu a experimentální ověření jeho vlastností je tak jasným cílem pro budoucí práci na tomto tématu.

Předposlední kapitola je věnována popisu směřování mojí disertační práce, přičemž hlavní téma tohoto článku představuje první krok na cestě k cíli vytyčenému v rámci disertace.

Poděkování

Tato práce byla podpořena Evropským fondem regionálního rozvoje (ERDF) v rámci projektu Centra excelence IT4Innovations (CZ.1.05/1.1.00/02.0070) a částečně také grantem FIT-S-11-1 – Pokročilé bezpečné, spolehlivé a adaptivní IT.

Reference

[1] INTEC Inc.: IPv4 Exhaustion Counter (English) [online]. September 2011. Available at URL: `http://inetcore.com/project/ipv4ec/index_en.html` (June 2012).

[2] IEEE Computer Society: Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications; Amendment 4: Media Access Control Parameters, Physical Layers, and Management Parameters for 40 Gb/s and 100 Gb/s Operation. IEEE std 802.3ba-2010, June 2010. ISBN 978-0-7381-6322-2.

[3] W. Eatherton, G. Varghese, and Z. Dittia: Tree Bitmap: Hardware/Software IP Lookups with Incremental Updates. ACM SIGCOMM Computer Communications Review, 34(2): 97–122, 2004.

[4] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink: Small Forwarding Tables for Fast Routing Lookups. In Proceedings of the ACM SIGCOMM '97, September 1997, pp. 3–14.

[5] Xilinx, Inc.: Xilinx Zynq-7000 Extensible Processing Platform [online]. June 2012. Available at URL: `http://www.xilinx.com/products/silicon-devices/epp/zynq-7000/` (June 2012).

[6] V. Puš, J. Tobola, J. Kaštil, V. Košař, and J. Kořenek: Netbench - the Framework for Evaluation of Packet Processing Algorithms. In Proceedings of the 7th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2011, pp. 95–96. ISBN 978-0-7695-4521-9.

[7] Xilinx, Inc.: Virtex-5 Family Overview. DS100 (v5.0), February 2009.

[8] Xilinx, Inc.: Virtex-6 Family Overview. DS150 (v2.4), January 2012.

[9] Xilinx, Inc.: 7 Series FPGAs Overview. DS180 (v1.11), May 2012.