OPTIMIZATION OF VOICE RECOGNITION FOR MOBILE DEVICES

Martin Tomec
Master Degree Programme (2), FIT BUT
E-mail: xtomec05@stud.fit.vutbr.cz

Supervised by: Petr Hanáček
E-mail: hanacek@fit.vutbr.cz

ABSTRACT

This paper deals with optimization of keyword spotting algorithm on processor architecture ARM Cortex-A8. It describes this architecture and especially the NEON unit with SIMD instructions. It also briefly describes keyword spotting algorithms used for implementation. Finally there are sumarized optimizations of keyword spotter for ARM Cortex-A8 architecture and their impact on performance.

1 ÚVOD

Rozpoznávání řeči strojem je značně náročné na výpočetní výkon a ještě před několika lety by bylo nemyslitelné používat k němu mobilní zařízení. Výkon nízkopříkonových procesorů se však dnes pohybuje v řádech miliard operací za sekundu, takže na mobilních platformách lze implementovat detektor klíčových slov. Jde sice o výrazně jednodušší úlohu, než je plné rozpoznávání plynulé řeči, ale pro tato zařízení ji lze s výhodou využít pro hlasové ovládání nebo pomalé diktování textu.

V mobilních zařízeních dnes dominují procesory založené na architektuře ARM, zejména díky svému nízkému příkonu. Tento článek se zaměřuje konkrétně na verzi ARMv7, kterou v současné době využívá většina nových mobilních telefonů. Cílem práce tedy není optimalizace algoritmů jako takových, ale optimalizace výsledného kódu pro danou architekturu.

2 ARCHITEKTURA ARM

Významným rozšířením u architektury ARMv7 je aritmetická vektorová jednotka NEON, určená pro podporu zpracování objemných dat (například multimediálních). Tato jednotka je oddělená od jádra procesoru, má vlastní load/store jednotku (tedy vlastní nezávislý přístup do paměti cache) a vlastní sadu registrů. Díky tomu může zpracovávat instrukce téměř nezávisle na jádru procesoru a pouze vydávání instrukcí do této jednotky je závislé na jádru procesoru. Je založena na architektuře typu SIMD a umožňuje tak jednou instrukcí provádět výpočet nad více hodnotami paralelně. Pracuje s šířkou slova 128 bitů, takže jedna instrukce zpracuje například čtyři 32-bitová čísla. Rozdělení slova však může být téměř libovolné. Její hlavní výhodou je však možnost zřetězeného zpracování instrukcí, která se uplatní zejména u zpracování vektorů.

Pokud mezi instrukcemi nejsou datové závislosti, mohou se průběhy zpracování jednotlivých instrukcí překrývat v čase. Takto lze teoreticky zpracovat v každém taktu dvě instrukce, avšak pouze některé kombinace typů instrukcí lze vydávat souběžně po dvojicích(viz [2]).

K hlavním nevýhodám jednotky NEON patří omezení čísel na jednoduchou přesnost a omezená podpora překladačů. I když většina překladačů podporuje automatickou vektorizaci kódu, není tato technika optimální a ruční vektorizací lze dosáhnout znatelného zrychlení. Pro představu jsou v tabulce 1 výsledky optimalizace jednoduché smyčky pro součet dvou vektorů. Ta byla nejprve přeložena bez vektorizace, pak s automatickou vektorizací překladače a nakonec ručně optimalizována pomocí vestavěných funkcí překladače (pro generování příslušných instrukcí jednotky NEON). Pro všechny překlady byl použit překladač GNU C++ verze 4.4.1. V tabulce je uveden průměrný čas pro 5 běhů programu. Do doby běhu je započítán pouze uživatelský čas, tzn. položka user z výstupu nástroje time.

Vektorizace	Žádná	Automatická	Ruční
Doba běhu – celočíselně	4,08 s	2,98 s	1,84 s
Doba běhu – jednoduchá přesnost	22,77 s	3,05 s	1,91 s

Tabulka 1: Porovnání doby běhu programu, podle typu vektorizace

Obrovský výkonostní propad (ze 3 s na 23 s) u čísel s plovoucí řádovou čárkou je způsoben tím, že překladač používá jednotku NEON pouze jako koprocesor pro výpočet v plovoucí řádové čárce. Data tedy nejsou načítána z paměti přímo do jednotky NEON, ale vždy přes jádro procesoru. Přenos dat mezi těmito jednotkami trvá 20 taktů a během zpětného přenosu (do jádra procesoru) nelze vykonávat žádné instrukce pracující s hlavní registrovou sadou (viz [2]). I z tohoto důvodu je vhodné v překladači nastavit vektorizaci kódu.

3 DETEKTOR KLÍČOVÝCH SLOV

Protože cílem této práce není optimalizace na úrovni algoritmů, byl pro optimalizaci vybrán již implementovaný detektor z diplomové práce Tomáše Cipra[1]. Tento detektor byl navržen s ohledem na možnosti mobilních zařízení a není tedy příliš náročný na výpočetní výkon. Podrobnější popis detektoru lze nalézt v původní práci. Zde bych shrnul pouze základní vlastnosti:

- Vstupní signál s vzorkovací frekvencí 8 kHz se dělí na rámce o velikosti 25 ms a rozestupu 10 ms.
- Z každého rámce se získává metodou perceptuální lineární predikce 13 koeficientů.
- Pro detekci fonémů používá neuronovou síť s jednou skrytou vrstvou, která obsahuje 100 neuronů.
- Pro následnou detekci slov byl použit Viterbiho dekodér.

Zdrojový kód detektoru bylo nutné upravit, protože používal knihovny specifické pro operační systém Symbian.

4 POSTUP OPTIMALIZACE

Po úpravě zdrojového kódu byl implementovaný detektor testován přímo na vývojovém kitu s procesorem OMAP 3530, kde byly pomocí profilování vybrány funkce vhodné pro optimalizaci. Bez automatické vektorizace byly výsledky téměř shodné s klasickou architekturou. Přibližně polovinu doby výpočtu tvořilo vyhodnocení neuronové sítě a třetinu tvořil výpočet Fourierovy Transformace (FFT). Po zapnutí automatické vektorizace však naopak funkce FFT tvořila polovinu doby výpočtu. Proto byla pro výpočet FFT použita knihovna FFTW. Ta sice zatím nepodporuje výpočty v jednotce NEON, ale i přesto dosahuje lepších výsledků díky optimalizaci algoritmu. Další optimalizace se zaměřila na ruční vektorizaci výpočtu neuronové sítě. V době psaní příspěvku byla dokončena jen částečně, proto jsou níže uvedené výsledky pouze předběžné. Cyklus pro výpočet výstupní vrstvy byl ručně vektorizován s použitím vestavěných funkcí překladače a ostatní cykly byly zjednodušeny, aby mohly být automaticky vektorizovány.

5 VÝSLEDKY

Původní verze zpracovala záznam o délce 60 sekund v průměru za 23,3 s. S použitím automatické vektorizace trvalo zpracování 22,6 s. V celém programu se totiž překladači podařilo vektorizovat pouze 5 smyček. Použitím knihovny FFTW se celková doba běhu snížila na 20,1 s. Konečnou ruční vektorizací se podařilo dosáhnout času 18,3 s. Výsledky jsou shrnuty v tabulce 2, kde je uvedeno i odpovídající vytížení procesoru, pokud by detektor běžel v reálném čase. Metodika měření byla stejná jako v kapitole 2.

Vektorizace	Žádná	Automatická	Automatická	Automatická i ruční
FFT	Původní	Původní	FFTW	FFTW
Doba běhu	23,3 s	22,6 s	20,1 s	18,3 s
Využití procesoru	38,8 %	37,7 %	33,5 %	30,5 %

Tabulka 2: Doba běhu programu podle použitých optimalizací

6 ZÁVĚR

Uvedený detektor klíčových slov lze použít na nových mobilních procesorech i k detekci v reálném čase. Optimalizací bylo dosaženo snížení zátěže procesoru při online rozpoznávání z původních 38,8 % na výsledných 30,5 %. Zbývající výkon mohou využít další aplikace, nebo lze snížit taktovací frekvenci procesoru. Vzhledem k výkonové rezervě lze uvažovat o rozšíření počtu koeficientů vstupujících do neuronové sítě, čímž by se zvýšila přesnost detekce.

REFERENCE

[1] Cipr, T.: Implementace detektoru klíčových slov do mobilního telefonu (Symbian 60). FIT VUT v Brně, 2007, diplomová práce.

[2] Cortex–A8 Technical Reference Manual [online]. Revision: r3p2. 2009 [cit. 2010-02-28] Dostupný na: <http://infocenter.arm.com/>

[3] Szöke, I.; Schwarz, P.; Burget, L.; aj.: Phoneme based acoustics keyword spotting in informal continuous speech [online]. 2005 [cit. 2010-01]