Spracovanie obrazu v reálnom čase, softvér (2)

Programovacie jazyky reálneho času

Programovacie jazyky reálneho času môžeme rozdeliť na tri druhy, a to na univerzálne na vysokej úrovni programovacích jazykov, hardvérové jazyky a nízko úrovňové symbolické jazyky. Najpoužívanejší programovací jazyk na spracovanie obrazu v reálnom čase na existujúcich procesoroch je C/C++. C je populárny hlavne preto, že každý procesor má kompilátor, ktorý môže zostaviť C zdrojový kód do natívneho strojového kódu. Pre túto výhodu možno nájsť veľa knižníc na spracovanie obrazu, ktoré sú určené pre rôzne cieľové oblasti. C má tiež výhodu v manipulácii s bitovými operáciami a schopnosť ovládať objekt alebo pole objektov v prípade použitia C++. Je to výhoda pre spracovanie obrazu v reálnom čase [1]. Preto sa po navrhnutí prototypu v LabVIEW alebo MATLAB-e odporúča, aby sa programoval algoritmus v C ako referencia, ktorá bude neskôr použitá na špecifickej platforme, pravdaže ak hardvérová platforma podporuje FPGA (Field-Programmable Gate Array). Pre FPGA existuje mnoho popisných hardvérových jazykov (HDLs – Hardware languages destription), veľmi populárny je VHDL (VHSIC hardware description language, VHSIC – Very-High-Speed Integrated Circuits). Programovanie v VHDL je často neznámou pre vývojára algoritmu, nakoľko vyžaduje podstatne iný štýl programovania. Nástroje MATLAB podporujú transformáciu systému navrhnutého v MATLAB-e priamo na implementáciu do FPGA [10]. Tieto nástroje tak prispievajú k rozširovaniu výhod riešení založených na FPGA. Ak sa nám v procese spracovania objaví hrdlo a nedokážeme vykonať optimalizáciu, možno tak spraviť v nízko úrovňovom programovacom jazyku, čo však vyžaduje hlbokú znalosť architektúry procesoru a ide o zdĺhavý proces.

2.1.2 Návrh softvérovej architektúry

Prekladania zdrojového kódu z výskumného prostredia do prostredia reálneho času je komplikovaná úloha. Vzhľadom na to, že spracovanie obrazu v reálnom čase sa obyčajne skladá z tisícky riadkov kódu, je potrebné, aby sa od začiatku projektu dodržovali princípy návrhu, čo zabezpečuje prehľadnosť, možnosť zmien algoritmov a flexibility v prípade úprav hardvéru [12]. Bez riadnej štruktúry by hrozilo riziko, že systém skončí ako neprehľadná zbierka kódov. Jednou z kľúčových metód je vytvoriť modulárny systém, ktorý nezahrnuje detaily, ale má definované rozhrania a spôsob komunikácie medzi blokmi. Takýto spôsob riešenia prináša výhodu v jednoduchosti vymieňania blokov [13]. Tiež je výhodné vytvoriť vrstvy alebo úrovne spracovania, kde existujú rozhrania medzi úrovňami a v prípade úpravy hardvéru treba upraviť len úroveň riadiacu hardvér.

2.1.3 Operačné systémy reálneho času

Pri vykonávaní úloh v reálnom čase treba uvažovať o rôznych stupňoch reálneho času, ako je spomenuté v prvom článku (tvrdý, mäkký reálny čas). Úlohou operačného systému v reálnom čase je prideliť priority jednotlivým úlohám a na základe priorít spracovať jednotlivé úlohy. Pre prenosné zariadenia, ako je digitálna kamera, operačný systém reálneho času môže používať rozličné aplikácie, ako je riadenie časového nastavenia, plánovanie úloh alebo správa vstupno-výstupných operácií [9]. Preto je operačný systém na prácu v reálnom čase dôležitým komponentom softvéru spracovania obrazu a videa v reálnom čase. Operačný systém garantuje dodržanie potrebných časov spracovania, a tak do istej miery zaisťuje deterministické správanie.

2.2 Správa pamäte

Ročné zvyšovanie pamäťového výkonu pomaly zaostáva za zvyšovaním výpočtového výkonu. Zvyšovaním kvality spracúvaného signálu narastá aj potreba väčšej pamäte na spracovanie obrazu v reálnom čase. Kvôli veľkému významu správy pamäte táto časť pokrýva kľúčové koncepty, ako základy pamäťovej architektúry, ukladanie obrazových dát a niekoľko optimalizačných stratégií správy pamäte.

2.2.1 Výkon pamäte

Kvôli stále rastúcej medzere vo výkone medzi výpočtovým a pamäťovým výkonom sa stáva optimalizácia pamäte viac kritická ako optimalizácia výpočtov, vzhľadom na väčšinu algoritmov, kde je na prvom mieste obmedzenie pamäte a až potom obmedzenie výpočtového výkonu [7]. Dôležité je vyhľadať metódy, ako optimalizovať dávky idúce do vyrovnávacej pamäte, aby sa preniesol maximálny objem obrazových údajov, ktoré budú potrebné na spracovanie v pamäti procesora. Samozrejme ďalší užitočný nástroj pre účinný pohyb údajov naprieč systémom bez zaťaženia CPU je DMA (direct memory acces – priamy prístup do pamäte).

2.2.2 Hierarchia pamäte

Najmodernejšie hardvérové architektúry sú vybavené hierarchiou pamätí, kde je každá úroveň oddelená. Čím je úroveň ďalej od procesora, tým väčšia je kapacita pamäte a tým väčší je aj čas prístupu. Zvyčajne hierarchia zahrnuje dve vyrovnávacie pamäte úrovne L1 a L2 nasledované vonkajšou pamäťou s pomerne vysokou dynamikou. L1 je umiestnená najbližšie k jadru CPU a obvykle je menšia ako L2, ktorá je ďalej od CPU ako L1. Niektoré architektúry umiestňujú tieto pamäte priamo na procesor, zatiaľ čo iné ich majú fyzicky oddelené. Napríklad v procesoroch (DSP) TMS320C64x je oddelená L1 programovateľná údajová vyrovnávacia pamäť a L2 vyrovnávacia pamäť. Pamäť dostupná na procesore poskytuje najrýchlejší prístup medzi rôznymi typmi pamätí.

2.2.3 Usporiadanie obrazových údajov v pamäti

Na základe častého využívania pamäte pri spracovaní obrazu v reálnom čase je vhodné vedieť, ako sú obrazové údaje reprezentované v C/C++ a aké sú bežné metodiky na ich sprístupnenie. Obrazové údaje sa ukladajú v riadkovom formáte, to znamená, že obrázok je uložený ako pole riadkov na rozdiel od MATLAB-u, ktorý údaje ukladá v stĺpcovom formáte ako pole stĺpcov. C/C++ využíva indexáciu poľa od 0 a hranaté zátvorky ako spôsob indexácie poľa, kým v MATLAB-e sa začína indexácia od 1 a pre indexáciu využíva oblé zátvorky. Pre vyššiu rýchlosť nie je vhodnejšie ukladať údaje do dvoj- alebo trojrozmerného poľa, ale do riadkov. Ak sa teda bude ukladať obraz, ktorý je dvojrozmerný MxN, malo by sa k nemu pristupovať $[N \cdot i + j]$ namiesto $[\,i\,][\,j\,]$ alebo pre trojrozmerný obraz MxNxL by sa malo použiť $[N \cdot L \cdot i + L \cdot j + k]$ a nie $[\,i\,][\,j\,][\,k\,]$.

2.2.4 Priestorové miesto a vyrovnávacia pamäť

Keď CPU číta alebo zapisuje na pamäťové miesto v hlavnej pamäti, najskôr skontroluje, či je dané pamäťové miesto vo vyrovnávacej pamäti. Ak CPU nájde dané pamäťové miesto vo vyrovnávacej pamäti, hovoríme tomu „cache hit“, inak hovoríme o „cache miss“. Pomer prístupov, ktorých výsledkom bol „cache hit“ ku „cache miss“ nazývame „hit rate“ a je meradlom efektívnosti vyrovnávacej pamäte.

Riadkové ukladanie znamená, že obrazové elementy, ktoré sú navzájom pri sebe v 2D, nie sú takto pri sebe uložené aj v pamäti [8], [7]. Vodorovne susediace obrazové elementy $[\,i\,][\,j\,]$ a $[\,i\,][\,j+1\,]$ sú oddelené len pár bitmi v pamäti, zatiaľ čo zvisle susediace obrazové elementy $[\,i\,][\,j\,]$ a $[\,i+1\,][\,j\,]$ sú oddelené niekoľkými bajtmi [8]. Následkom toho sa riadkové ukladanie obrazových údajov uprednostňuje pred stĺpcovým. V prípade stĺpcového ukladania dochádza k častejšiemu „cash miss“, a teda k neefektívnemu využívaniu vyrovnávacej pamäte.

2.2.5 Stratégie optimalizácie pamäte

Vnútorná verzus vonkajšia pamäť
Pre rýchlejšie prístupy je vhodnejšie, ak je pamäť priamo na CPU, preto by bolo vhodné, aby sa celý spracúvaný obraz dostal do pamäte na čipe, čo je však veľakrát nemožné. Jedna zo stratégií je rozdelenie údajov do blokov a následné spracovanie údajov po blokoch. Niektoré dôležité obrazové údaje rozdeľuje schéma rozdelenia zahrnujúca aj riadkové rozdeľovanie, ktoré je najpoužívanejšie. Riadkové rozdelenie rozdelí obraz do riadkov, ktoré sú následne prenesené do vnútornej pamäte na rýchlejšie spracovanie dát. Dôsledkom prenesenia časti obrazu potrebného na spracovanie je znižovanie „cash miss“. Ďalšou možnosťou je rozdelenie do blokov, ktoré sa navzájom prekrývajú v závislosti od typu práve vykonávanej operácie. Avšak do vnútornej pamäte by sa mali dostať aj často používané informácie [14].

Efektívny presun údajov
Pri realizácii účinného používania vnútornej pamäte na ukladanie obrazových údajov je dôležitý efektívny presun údajov, avšak používať CPU na vykonanie presunu, napríklad cez funkciu memcpy, nie je vhodné [4], [5]. Kľúčom je dostupnosť technológie DMA, kde procesor len inicializuje prenos a následne môže vykonávať spracovanie obrazu a nie sa zaoberať prenosom dát. DMA umožňuje riadiť viaceré kanály súčasne, čo znamená, že môže poskytovať viacnásobný prenos dát. S dostupnosťou DMA a účinnou stratégiou multibuffer je možné paralelné presúvanie údajov. Pomocou paralelného spracovania a multibuffer vieme docieliť nezávislé alebo závislé paralelné spracovanie údajov. V závislosti od typu spracovania sa zvyčajne používajú tri buffre, kde buffer 1 a buffer 2 pracujú v „ping-pong“ správe údajov a buffer 3 je výstupný. DMA využíva buffer 1 na skladovanie údajov pripravených na spracovanie, zatiaľ čo buffer 2 sa využíva pri spracovaní a v bufferi 3 sú umiestnené výsledky. Po spracovaní údajov v bufferi 2 sa výsledky z buffera 3 posielajú na vonkajšiu pamäť cez DMA kanál, zatiaľ sa spracúvajú údaje v buffer 1 a ďalšie údaje prichádzajú do buffera 2 cez DMA kanál. Je viac spôsobov, ako využívať jednotlivé buffre. DMA prenosmi môžeme docieliť maximálne využívanie zbernice [16], [17].

Zrýchlenie prístupu do pamäte prostredníctvom spôsobu spracovania
Ďalšia metóda redukujúca spomalenie vonkajšej pamäte pri spracovaní obrazu je založená na zmene spôsobu spracovania obrazu, a to z operácií využívajúcich celý obraz na vstupe a výstupom je jeden obrazový element, na operácie, kde vstupom aj výstupom je jeden obrazový element. Pri operáciách, kde vstupom a výstupom je jeden obrazový element, možno použiť viacnásobné spracovanie na obrazových údajoch, pretože sa tam nenachádzajú žiadne údajové závislosti [15]. Operácie založené na spracovaní obrazu sú také, ktoré používajú jednu operáciu na všetky obrazové elementy a potom ďalšiu operáciu na všetky obrazové elementy. Toto je podobný spôsob spracovania ako MATLAB. Operácie založené na spracovaní obrazového elementu sú také, ktoré všetky operácie vykonajú na jednom obrazovom elemente, a to sa následne opakuje pre všetky obrazové elementy. Problémom operácií založených na spracovaní obrazu je to, že nedokážu efektívne využívať pamäť, lebo treba čítať ten istý obrazový element viackrát z externej pamäte pre úplné dokončenie. Operácie založené na spracovaní obrazového elementu načítajú ten istý element len raz z externej pamäte. A tak aj operácie, ktoré sú určené na spracovanie obrazového elementu, pomáhajú zvýšiť výpočtový výkon implementácie. Ten je definovaný ako percento počtu inštrukcií vykonaných k počtu prístupov do pamäte. Ak je pomer vykonaných inštrukcií k počtu prístupov do pamäte veľký, vieme, že dané riešenie má dobrý výpočtový výkon. Ak je tento pomer malý, ide o nízky výpočtový výkon, čo znamená, že daná rutina je výpočtovo neefektívna.

Ostatné optimalizačné metódy
Je mnoho ďalších stratégií, ktoré by mohli byť použité na účinné využívanie pamäte, a všetky závisia od aplikácií, dostupných zdrojov a kritických obmedzení pri spracúvaní. Existujú rôzne kroky, ako používať globálnu premennú namiesto lokálnej premennej, prideľovanie pamäte pre hromadné dynamické prideľovanie, využívanie prenosu komprimovaných údajov.

Literatúra

(vybrané tituly)

[1] ACKENHUSEN, J.: Real-Time Signal Processing: Design and Implementation of Signal Processing. NJ: Prentice-Hall: Englewood Cliffs, 1999

[4] PATEL, K.: Porting PC Based Algorithms to DSPs. 2003, Embedded Edge

[5] PATEL, K.: Porting and Optimization Techniques for C++ Based Image Processing Algorithms on TMSC62x DSP, TI Developer Conference. : 2004

[7] H. CHENK, LI, WEI, B.: Memory Performance Optimizations for Real-Time Software HDTV Decoding. : September 2005

[8] CHARLES, BRIFAULT: Data Cache Management on EPIC Architecture: Optimizing Memory Access for Image Processing, ACM SIGGARCH Computer Architecture News. : 2004

[9] KAO, T., SUN, W., LIN, S.: A Robust Embedded Software Platform for Versatile Camera Systems Proceedings of the IEEE International Symposium on Circuits and Systems. : 2005

[10] MATLAB DSP Algorithmic Synthesis for FPGAs and ASICs. www.accelchip.com

[11] Catalytic: Catalytic MCS Family: MATLAB to C Code Synthesis. www.catalyticinc.com

[12] SANGWAN, R. R., LUDWIG, LAPLANTE, P., NEILL, C.: Performance Tuning of Imaging Applications Through Pattern-Based Code-Transformation, Proceedings of SPIE-IS&T Electronic Imaging Conference on Real-Time Imaging. : 2005

[13] LOGANTHAN, V.: Flexible and Scalable Movie Architecture for DSP Based DSC/DM Systems. http://www.technonline.com/

[14] JARA, P., GARCIA, I., USEVITCH, B.: Analysis and Optimization of JPEG2000 in the TMS320C6701, Proceedings of the Global Signal Processing Conference and Expo. : 2003

[15] BRAMBERGER, M., BRUNNER, J., RINNER, B., SCHWABACH, H.: Real-Time Video Analysis on an Embedded Smart Camera for Traffic Surveillance, Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium. : 2004

[16] LARIN, S.: Introduction to AltiVecTM Technology: Ten Easy Ways to Vectorize Your Code, Freescale Technology Forum. : 2005

[17] MANAGULI, R., YORK, G., KIM, D., KIM, Y.: Mapping of Two-Dimensional Convolution on Very Long InstructionWord Media Processors for Real-Time Performance. : 2000

Pokračovanie v budúcom čísle.

Ing. Tomáš Surovčík
prof. Ing. Ladislav Jurišica, PhD.

Slovenská technická univerzita v Bratislave
Fakulta elektrotechniky a informatiky
Ústav riadenia a priemyselnej informatiky
Ilkovičova 3, 812 19 Bratislava
e-mail: tomas.surovcik@stuba.sk