Spracovanie obrazu v reálnom čase, softvér (3)

2.3 Optimalizácia softvéru

Ak máme kód napísaný v C štandarde a vykonali sme jeho implementáciu na cieľovom hardvéri s využitím optimalizácie prekladača pri kompilácii kódu, nastáva čas na vykonanie softvérovej analýzy. Pri softvérovej analýze treba vedieť, aké časti kódu vyžadujú najväčší výpočtový výkon v celom procese spracovania obrazu v reálnom čase, teda ktoré časti kódu nám tvoria hrdlo v celom procese spracovania. Využívanie knižníc na spracovanie obrazu, aritmetika s pevnou desatinnou čiarkou, softvérové prúdové spracovanie a paralelizmus sú pri spracovaní obrazu využívané najčastejšie.

2.3.1 Analýza

Ide o stanovené metódy softvérovej optimalizácie na zlepšenie efektivity zdrojového kódu. Nemalo by sa postupovať naslepo, ale podľa stanovených pravidiel. Na prvom stupni softvérovej optimalizácie je analýza kódu, čo znamená zhromaždenie informácií o počte taktov počas vykonávania jednotlivých funkcií kódu. Možnosti analýzy ponúkajú aj najmodernejšie vývojové prostredia. Analýza kódu odhaľuje časti, ktoré sú najviac zdĺhavé. Cieľom softvérovej optimalizácie je zmeniť kód tam, kde je to časovo najkritickejšie, a tak dosiahnuť prijateľný výkon na základnej hardvérovej architektúre. Časovo kritické časti sa väčšinou nachádzajú vo vnorených cykloch. Po úprave kódu by sa mala overiť jeho funkčnosť a následne opakovať analýza. Ak sa ani teraz nedosiahnu uspokojivé výsledky, treba celý proces opakovať. V prípade dosiahnutia uspokojivého výkonu môže byť optimalizácia zastavená.

2.3.2 Optimalizácia na úrovni kompilátora

Moderné kompilátory sú vybavené automatickou softvérovou optimalizáciou Získavajú správu z kódu a pokúšajú sa ho zmeniť tak, aby bol účinnejší. Často je tam niekoľko úrovní optimalizácie a cieľom každej je rôzny aspekt optimalizácie. Po použití optimalizácie prekladačom by malo byť zjavné zvýšenie výkonu. Niekedy sa však môže stať, že kompilátor zníži výkon konkrétnej funkcie namiesto jeho zvýšenia. Preto je dobré aplikovať optimalizáciu prostredníctvom kompilátora len na tie funkcie, ktoré zaznamenajú zvýšenie výkonu.

2.3.3 Pevná verzus pohyblivá desatinná čiarka

Pohyblivá alebo plávajúca desatinná čiarka znižuje výkon v reálnočasových implementáciách na procesoroch s pevnou desatinnou čiarkou, ktoré sa často používajú v prídavných zariadeniach. Pre procesor s pevnou desatinnou čiarkou sa počítanie s plávajúcou desatinnou čiarkou pokladá za plytvanie výkonu, pretože to treba riešiť na softvérovej úrovni. Vzhľadom na to, že zložité operácie sa vykonávajú v cykloch, softvérové simulovanie desatinnej čiarky by znížilo výkon o to výraznejšie. Optimalizácia kompilátora obvykle nefunguje správne v cykloch s pohyblivou desatinnou čiarkou na procesore s pevnou desatinnou čiarkou a tiež neoptimalizuje cykly s volaním funkčných procedúr vnútri cyklu [4], [5]. Počítanie s pevnou desatinnou čiarkou je preferovaný formát pre aritmetické výpočty zvlášť v systémoch s reálnym časom. Dôvod preferovania pevnej desatinnej čiarky pred pohyblivou je ten, že spracovanie je rýchlejšie. Zrýchlenia počítania pri prechode z pohyblivej desatinnej čiarky môžu byť dramatické, a preto sa považuje počítanie s pevnou desatinnou čiarkou za základ na dosiahnutie vyššieho výkonu. Avšak jedna vec je o tomto probléme diskutovať a iné prikloniť sa k realizácii počítania cez pevnú desatinnú čiarku, nakoľko sa treba rozhodnúť, aký pevný formát je vhodný pre výpočty. Jedna z najznámejších metodík kódovania pevnej desatinnej čiarky je reprezentovať desatinné číslo ako zmiešaný zlomok Q formát. Tento formát umožňuje určiť úroveň presnosti tak, aby sa potlačil efekt kvantovania. Následkom toho je dôležité určiť úroveň presnosti pre každý výpočet a potom vhodne zvoliť počet desatinných miest.

2.3.4 Optimalizované softvérové knižnice

Knižnice sú vyvíjané na dosiahnutie maximálneho výkonu pre danú architektúru počítača a tak sa ich používanie považuje za prostriedok na zvýšenie výkonu. Pre implementáciu knižnice treba naštudovať dokumentáciu funkcií danej knižnice. Ak sa používajú knižnice, nielenže to skráti vývoj, ale nie je potrebná optimalizácia základných funkcií, ktoré sa často vykonávajú. Zatiaľ čo sa knižnice používajú na zvýšenie výkonu, v určitých prípadoch sú používané na špecifické zamerania a rátanie špecifických úloh namiesto využitia univerzálnych funkcií [18]. Knižnice na spracovanie obrazu:

- Knižnica Gandalf
- Knižnica IM
- Knižnica Image Magick
- Knižnica ITK
- Knižnica Matrox Imaging Library
- Knižnica OpenCV

2.3.5 Predpripravené informácie

Niekedy sa výpočty opakujú a vtedy možno výsledok považovať za konštantu, ktorá sa v danom cykle nemusí počítať. V takomto prípade je vhodné takéto konštanty ukladať v tabuľke a postupom času ich sprístupniť ako konštanty v pamäti. Často používané konštanty by mali byť vo vnútornej pamäti, ak to priestor nedovoľuje, je potrebné, aby boli v rýchlej externej pamäti. Ak ide o konštanty používané len za určitých okolností, bolo by užitočné ich uložiť do vnútornej pamäte len v prípade, ak nastanú okolnosti, pri ktorých sú dané konštanty potrebné.

2.3.6 Podprogramy verzus vnorený kód

Na zvýšenie výkonu sa odporúča používať vnorený kód namiesto podprogramu [1]. Podprogramy na jednej strane sprehľadňujú kód a uľahčujú programovanie, na strane druhej ak sa zavolá podprogram, premenné, ktoré sa momentálne nachádzajú v pamäti, sa musia odložiť a po dokončení znova vybrať. Podprogram možno nahradiť vnoreným kódom, následkom čoho sa zväčší veľkosť kódu a zrýchli sa počítanie.

2.3.7 Vetva predspracovania

Ide o kódy, ktoré nemožno paralelizovať v ich pôvodných formách. Jediným riešením pre zrušenie tohto obmedzenia je použiť predspracovanie [16], [3].

2.3.8 Zmena cyklov

Cykly sú zvyčajne najkritickejšie časti kódu. Na ich optimalizáciu je veľa stratégií, ktoré majú priniesť zvýšenie výkonu. Často sa najskôr pristupuje k zmene cyklu na vyššej programovej úrovni a až potom sa používa nižšia úroveň (jazyk symbolických inštrukcií). Prvým krokom pri transformácii cyklu je preskúmanie. Tu treba zistiť, či sa v cykle nenachádzajú zbytočné výpočty. Ak sa zistia zbytočné výpočty vnútri cyklu, treba ich odstrániť. Za zbytočné výpočty možno považovať aj časť matematického výrazu, ktorá sa nemusí rátať v cykle a pre daný cyklus predstavuje konštantu. Tiež sa treba vyhnúť pohyblivej desatinnej čiarke na procesore s pevnou desatinnou čiarkou. Následne sa treba vyhnúť volaniu funkcií v tele cyklu, ak to nie je nevyhnutné, alebo ich nahradiť vnoreným kódom. Tu by sme mali poznamenať, že volanie C funkcií dodávaných výrobcom netreba vyhadzovať z cyklu, lebo tieto funkcie sa mapujú priamo na úrovni jazyka symbolických inštrukcií. Jednou z najbežnejších zmien cyklu je inštrukčná úroveň paralelizmu (ILP – instruction level parallelism). Rozbalenie cyklu je tiež jednou z metód, ako zrýchliť a zefektívniť kód. Tým, že cyklus rozbalíme, dovoľujeme, aby kompilátor lepšie vykonal plánovanie hlavného cyklu. Nevýhodou je zväčšenie kódu, a teda aj veľkosti, ale vyvažuje to rýchlejšie spracovanie. Softvérové prúdové spracovanie (pipelining) je technika optima-

lizácie zameraná na znižovanie času vykonania kritických cyklov, v podstate táto technika využíva IPL. Ak kompilátor nezabezpečí uspokojujúce výsledky prostredníctvom IPL, je možná ručná realizácia. To je však skutočne únavná úloha, nakoľko tento postup vyžaduje analyzovať údaje a uistiť sa, že neexistuje žiadna inštrukčná ani údajová závislosť v naplánovaných paralelných procesoch. Ručné plánovanie prináša však aj výhodu v podobe najväčšieho výkonu.

2.3.9 Komprimácia spracúvaných údajov (Packed Data Processing)

Obrazový element je často reprezentovaný ako 8-bitový, záleží však na požadovanej presnosti. Dnes sa používajú 32- a 64-bitové procesory, čo znamená, že môžeme spracovať štyri 8-bitové obrazové elementy naraz pri 32-bitovom procesore alebo 8 obrazových elementov pri 64-bitovom procesore. Tieto operácie sa môžu využívať na zrýchlenie maticových a vektorových operácií. Túto vlastnosť je vhodné využívať na to, aby sa spracovalo čo najväčšie množstvo údajov v jednom cykle; ak tento rys spojíme so softvérovým prúdovým spracovaním, možno dosiahnuť ešte lepšie výsledky.

Záver

Mnohé úlohy riadenia systémov vyžadujú využitie vizuálnych informácií na riadenie. Získanie a spracovanie obrazu v reálnom čase s využitím výsledkov na riadenie vyžaduje osobitne navrhnutý HW a SW. Pre náročné úlohy ide o riešenia s vysokou zložitosťou a cenou. Správne navrhnuté a implementované systémy však umožňujú dosiahnuť výsledky, ktoré sú inak nedosiahnuteľné.

Poďakovanie

Tento článok vznikol s podporou Grantovej agentúry MŠ VEGA pri riešení grantu VEGA 1/3120/06.

Literatúra

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

[2] KONIAR, D., HARGAŠ, L., HRIANKA, M.: Application of LabVIEW in Videosequence Processing, Applied Electronics 2007, Pilsen, 5. – 6. 9. 2007

[3] I. OZERT, LU, WOLF, W.: Design of a Real-Time Gesture Recognition System, IEEE Signal Processing Magazine. : 2005

[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

[6] PRESS, W., FLANNERY, B., TEUKOLSKY, S., VETTERLING, W.: Numerical Recipes in C: The Art of Scientific Computing. Cambridge, England: Cambridge University Press, 1992

[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

[18] KULKOVÁ, E: Prehľad knižníc na podporu spracovania obrazu. Bratislava: Fakulta matematiky, fyziky a informatiky Univerzity Komenského v Bratislave, 2007

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

61