УДК 519.245; 519.674; 004.032.24:004.272
DOI: 10.18372/2310-5461.37.12369

А. К. Шевченко**,** аспірант
Національний авіаційний університет
orcid.org/0000-0003-3863-0473
e-mail: lllandreyshevchenkolll@gmail.com

ВИБІР ПРОГРАМНО-АПАРАТНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ОПЕРАЦІЇ ЗГОРТКИ З ПІДВИЩЕНОЮ ШВИДКІСТЮ

Актуальність та постановка завдання

Базисом для процесу автоматичної векторізації програмного коду (ПК) є SIMD команди CPU, що використовуються сучасними компіляторами, як, наприклад GCC, і Clang/LLVM. Для цього, при компіляції програмного продукту (ПП), використовуються прапори компілятора, як, наприклад, -O3, або агресивніші -O4/-Ofast, наявність яких залежить від операційної системи та компілятора. Ці прапори дозволяють автоматично оптимізувати/векторизувати код ПП. Але, як було показано у статті [1, с. 78], цього недостатньо, прикладом цього може бути обробка цифрового зображення (ЦЗ) або потоку відео- (стабілізація, фільтрація, автокорекція шуму і т. п.). Слід врахувати деякі особливості вище перелічених завдань для ЦЗ, а саме:

- обчислювальна складність методу обробки ЦЗ;
- чи є/був оптимізований метод/алгоритм;
- апаратні можливості архітектури, на якій виконується завдання.

Виділимо найбільш ресурсомісткі, але і найбільш важливі, операції над ЦЗ: згортка, масштабування (як правило реалізоване за рахунок першої операції) і аналіз ЦЗ (його властивостей: кольору, контрасту, яскравості і т. п.). Операція згортки (далі СО — від англ. *Convolution Operation*) ЦЗ (1), є найбільш простою, але і у той же час, найбільш пріоритетною/ресурсомісткою задачею, з усіх перелічених вище:

$$p_{i,j} = \sum_{k=0}^{r}\sum_{l=0}^{c}\Gamma_{k,l}P_{k+i-a,l+j-a'} \qquad (1)$$

де $i = a\ldots, W-(r-a)-1$;

$j = a', \ldots, H-(c-a')-1$ — пікселі ЦЗ (відтінків сірого/кольорового ЦЗ); p_{ij} — операція лінійна фільтрація ЦЗ; (i, j) — індекси ЦЗ; W, H — розмір ЦЗ; P — складова растру джерела ЦЗ; p — складова растру джерела/приймача ЦЗ; Γ — матриця ядра згортки розміром $r \times c$.

Рівняння (1) є загальним і частково подібним до функції бібліотеки OpenCV- cv::filter2D(...) [2, с. 1]. Далі зазначимо, що будуть надані квадратні ядра Г для СО, тож $c = r$, і так буде надалі.

Для вирішення завдання зменшення часу виконання СО використовують програмно-апаратну архітектуру паралельних обчислень (ПАПО). Зауважимо, що до таких ПАПО відносять Nvidia CUDA (далі CUDA) і ATI Stream Technology (далі ATI – ST) для OpenCL API. Також, ці ПАПО є найпопулярнішими на момент публікування цієї статті. Виробники ПАПО надають достатній набір інструментів, а саме С-подібна мова та її модулі, у вигляді бібліотек і фреймворков. Основним інструментом ПАПО є сучасні GPU, що застосовуються для розпаралелювання будь-яких ресурсомістких завдань. Популярним GPU є відео карта GeForce GTX 1080 Ti, що значно зменшує час навчання CNN. Важливо відмітити, що шейдерні блоки GPU подібні мобільним CPU, які складають основу обчислювальної потужності сучасних телефонів і спроектовані на базі RISC.

Із зазначеного випливає, що підвищення продуктивності СО (1) призведе до загального підвищення продуктивності ПП мобільних платформ, одноплатних комп’ютерів і тому подібних систем, що в цілому широко використовують цю операцію, для робот із ЦЗ. Таким чином буде актуальним, підвищення продуктивності СО ЦЗ у: задачах фільтрації ЦЗ (корекції різкості, виділення країв, розмиття тощо); задачах масштабування ЦИ; задачах навчання класифікаторів/нейронних мереж; для мультимедійних задач в цілому (приміром обробка відео і т.п.). Крім того, зменшення часу виконання СО, призведе до зменшення часу навчання CNN, що на момент написання статті є актуальним.

Метою даної роботи є огляд програмно-апаратних засобів, що дозволяють виконувати операцію СО (та подібні до неї) за найменшу/найшвидшу одиницю часу. Це дозволить найефективніше (у подальших дослідженнях) обрати архітектуру та методику проектування нового методу СО ЦЗ.

Також ця робота містить огляд альтернативних рішень (у вигляді бібліотек/фреймворків тощо) реалізації що мають реалізації функцій по обробці ЦЗ, з урахуванням різноманітних архітектур CPU/GPU. Крім того, надамо огляд програмних, апаратних та інших технологій та засобів, що прискорюють ПП як автоматично, так і за допомогою спеціальних прийомів/методів. Також в роботі буде надано зауваження, щодо використання експертної оптимізації.

Виклад основного матеріалу

Огляд сучасних підходів для оптимізації ПП розпочнемо з розгляду таких підтем, як: огляд апаратних засобів (апаратної архітектури), що надають максимальну продуктивність; огляд програмних засобів, що надають максимальну продуктивність; огляд сучасних обчислювальних підходів, що підвищують продуктивність.

Спочатку подамо огляд апаратних засобів, що надають максимальну продуктивність.

Огляд апаратних засобів (чи апаратної архітектури), за допомогою яких досягається максимальна продуктивність, розпочнемо з деякого узагальнення, яке надає/забезпечує підвищення продуктивності для CPU і GPU. Цим узагальненням є типи/види паралелізму, як на рівні даних, так і на рівні команд. Клас обчислювальних систем, що описують це узагальнення, закладений в класифікації/таксономії Флинна: SIMD, MIMD, MISD, SISD і в їх похідних [3, с. 1901]. Нижче приведена (табл. 1) розкриває сутність цієї таксономі:

Таблиця 1

Таксономія Фліна

Операції / Дані	Так	Ні
Так	MIMD	SIMD
Ні	MISD	SISD

Для CPU RISC/CISC архітектури, важливою є наявність SIMD у вигляді векторних команд мов асемблера. Популярним представником CISC архітектури є фірма Intel і її серія CPU x86. SIMD у ній представлений у вигляді розширень набору команд мови асемблера як SSE_n та $AVX_{1/2}$. Що ж до представників RISC архітектури CPU, то варто відмітити фірму ARM, що є основним сертифікатором/правовласником.

Найпопулярніші серії є Cortex-A8–23 та Cortex A53–72. SIMD представлений там у вигляді NEON32 і NEON64 розширення мови асемблера [4, с. 1200]. Окремо можна зауважити, що в сучасних CPU, MIMD не представлений (не має фізичного втілення у вигляді розширення мови асемблера), але частково присутня у сучасних GPU.

У сучасних GPU, для забезпечення паралелізму, використовують шейдерні блоки, що ґрунтуються на RISC архітектурі CPU (далі SCPU). Основою технології ПАПО є паралельне використання SCPU, що мають у наявність SIMD та MIMD. SIMD SCPU має регістри 128-bit, 256-bit або навіть більшої довжини. Також відмітимо, що кількість цих регістрів значно більша, ніж у сучасних CPU (більше ніж 32).

На жаль, SIMD/MIMD команди недоступні безпосередньо і фахівці використовують регламентований набір команд/інтрінсіків: бітові, логічні та інші операції. Що ж до програмної оптимізації для GPU, то слід розглянути ПАПО, які забезпечують паралелізм і використання SIMD та MIMD: CUDA та OpenCl. ПАПО OpenCl дозволяє використовувати CPU/DSP/FPGU, завдяки контекстно-незалежному підходу, що вигідно виділяє його від CUDA. Також варто відмітити, що MIMD принцип обчислень, досягається за рахунок використання паралельного використання векторних регістрів кожного шейдерного блока. Отже виробництво тієї, або іншої моделі GPU, спирається на вже існуючий пристрій від фірм, що сертифікують виробників GPU, а саме AMD/ATI та Intel/NVidia. Т.ч. якщо GPU відповідає аналогу фірми ATI, то ПАПО OpenCl являтиме програмну підтримкою API ATI-ST. Якщо ж GPU фірми NVidia, слід звернути увагу на API CUDA. Таким чином, GPU значно обходить за продуктивністю CPU, при комплексному навантаженні, прикладом якого може бути навчання CNN.

Слід окремо розглянути апаратне розширення сучасних CPU, у вигляді сопроцесорних модулів типу цифрового сигнального процесора (далі DSP, від анг. *Digital Signal Processor*). Фірма Qualcomm та її Snapdragon-625/635/845/835/825 CPU вбудовує (розроблені ними ж) DSP-Hexagon. Особливістю даного DSP є VLIW (very long instruction word) команд асемблера, що фактично означає мультипоточність на рівні команд асемблера. Це означає, що за одне переривання обробляється 3 команди асемблера з незалежними частинами даних (як SIMD, так і SISD). Це призводить до підвищення продуктивності алгоритмів, порівняно з SIMD оптимізацією (під NEON32 або NEON64), у 4 рази. Також, алгоритми оптимізовані під даний DSP, значно знімає навантаження з CPU і підвищує продуктивність 18 разів (при кодування/декодування відео/аудіо), а споживане навантаження зменшується на 75 % [5, с. 1]; [6, с. 1].

Далі розглянемо оптимізацію за допомогою програмних засобів. Процес розробки будь якого ПП спирається на такі інструменти як: компілятори/інтерпретатори, бібліотеки/фреймворки.

Наприклад, фреймворки, використовують всі перераховані можливості апаратних архітектур. Таким чином, розгляд сучасних підходів оптимізації, необхідно почати з огляду таких питань:

- за допомогою якого компілятора був створений даний ПП?
- за допомогою, яких бібліотек/фреймворків був розширений функціонал даного ПП?

Для сучасних компіляторів, важливим питанням є оптимальність авто-векторизації програмного коду. Від якості авто-векторизації залежить, чи буде цей ПП задовольняти користувачеві за критерієм швидкодії. В даному контексті, швидкодія залежить від можливостей компілятора оптимальним способом векторизувати ПП, не вносячи значного погіршення точності розрахунку. Тож, такі компілятори, як GNU Compiler Collection (далі GCC/G++) [7, с. 50], Clang [8, с. 45] та nvc (для компілювання CUDA *.cu файлів), дозволяють отримати якісний ПП.

Найбільш поширеним компілятором на сьогодні є GCC, розробкою/підтримкою якого займається FSF (співтовариство). GCC (gnu compiler collection) являє собою набір компіляторів для різних мов програмування та архітектур, що був розроблений у рамках проекту GNU Річардом Столлманом у 1985 р.

Clang ще один компілятор, що стає дуже поширеним і є головним конкурентом GCC. Корпорація Apple підтримує/використовує його, як базовий компілятор для своїх продуктів. Clang є фронтендом для мов програмування C, C++, Objective-C, Objective-C++ і OpenCL. Для оптимізації (авто-векторизації) і кодо-генерації в ньому використовується фреймворк LLVM.

З усього вище сказаного виходить, що розробники використовують GCC та Clang, для отримання оптимального вихідного ПП за критерієм швидкодії/якості. У статті [1, с. 78] було проведено порівняльний аналіз продуктивності експертної та автоматизованої векторізації СО для ЦЗ.

Було використано компілятор Clang (що входить до ndk-r14b) під OS Android 5.5.1(x64), CPU-MT6752(x64). Експериментально було доведено, що авто-векторизація компіляторів суттєво програє експертній, у якій використовувалися асемблерні вставки (далі Inline Assembly).

Що ж стосується компілятора nvcc, то він призначений для збільшення продуктивності ПП, за допомогою ПАПО CUDA. CUDA надає можливість реалізовувати алгоритми на спеціальному діалекті мови C, надалі використані на GPU фірми NVidia.

Окремо варто згадати OpenCl, що є єдиним стандартом, для розробки додатків гетерогенних систем, підтримуваний всіма вищезгаданими співтовариствами та фірмами. Зауважимо, що OpenCl це теж фреймворк, для написання комп'ютерних програм у вигляді сценарію, пов'язаних з паралельними обчисленнями, що відбуваються в момент запуску ресурсномісткою складової програми на GPU/DSP/CPU. Розробником OpenCl є The Khronos Group Inc.

При оптимізації ПП, мінусами ПАПО є уповільнення за рахунок швидкої зміни потоку даних, прикладом чого є потік даних з відеокамери. Причина проста — дуже великі накладні витрати при передачі одного кадру зображення на виконавчий пристрій (шейдерні блоки, тощо). Тому часто використовують принцип передачі даних у вигляді пакету 100–200 мегабайт, що призводить до значного зростання продуктивності (близько до 20 разів порівняно з CPU). Саме із-за цієї властивості, навчання сучасних CNN проводиться на GPU, де дані для навчальної вибірки можна підготувати і представити у вигляді готового пакета, що тривалий час буде доступній RAM виконавчого пристрою.

Далі розглянемо розширення можливостей ПП за допомогою фреймворків/бібліотек, що вільно розповсюджуються. Найчастіше, фреймворки створені на базі вже існуючих бібліотек, що дозволяє недоліки одних бібліотек перекрити перевагами інших. Це помітно підвищує якість вихідного ПП, створених за допомогою фреймворків, що і є перевагою фреймворків над бібліотеками. Що ж стосується переваг бібліотек, то під цим розуміється програмно-апаратна оптимізація, що містить всі вище описані підходи CPU/GPU/DSP/тощо. Найчастіше, програмною оптимізацією бібліотек є включення в себе SIMD векторізації ресурсномістких частин. У результаті досягається бажана оптимальна продуктивність бібліотек, що призводить до підвищення продуктивності фреймворків.

За останні кілька років частка SIMD оптимізації для різних архітектур RISC/CISC CPU суттєво зросла, а саме для сімейств CPU фірми ARM: armeaby-v7a і arm64-v8a. Open-Cv є гарним прикладом бібліотеки, що має у наявності SIMD оптимізацію ресурсномістких частин. Вона містить багатий набір оптимізованих алгоритмів, для різних архітектур CPU.

За рахунок перерахованих SIMD оптимізацій, OpenCV забезпечує підвищення продуктивності таких завдань: робота з ЦЗ (згортка, масштабування і т. п.); аналіз властивостей ЦЗ (контраст-

ність, яскравість і т. п.); також динамічно розвивається модуль для навчання CNN. Таким чином, для даного дослідження, бібліотека OpenCV є еталоном продуктивності та якості.

Окремо розглянемо ПП, комерційного та вільно типів. І для початку зазначимо один мало відомий факт, що у великих корпораціях, як, наприклад, Google, Apple, Microsoft і Oracl, є штат програмістів, які розробляють нові чисельні методи, які використовують "single point" арифметику, але результати, за критерієм точності подібні "floating point" алгоритмам. Це призводить до збереження точності і поліпшення продуктивності обчислювальних процесів.

Але безпосередньо сутність оптимізації (сутність, технологія і т. п.), є комерційною таємницею і недоступна широкому загалу, отже є слабо висвітленим у відкритих публікаціях. Тому, судити про ефективність будь-якого роду оптимізації комерційного ПП можна лише за його якістю, стабільністю і швидкодією порівняно з їх безкоштовними аналогами. Наведемо приклад підвищення продуктивності за допомогою SIMD оптимізацій, для комерційних продуктів: корпорація Microsoft та її ПП — ОС 'Windows 10', в якому використовується технологія векторизації $AVX_{1/2}$, для оптимізації ядра ОС; корпорація Oracle і її пропрієтарний ПП 'Oracle Java VM', у якому використовується SIMD оптимізація від Intel/AMD $AVX_{1/2}$/3DNow, що призводить до зростання продуктивності мови програмування Java та ПП, написаного за її допомогою.

Прикладом ПП, що є доступним для широкого загалу (під ліцензією GNU GPL), є серія ігрових рушіїв id Tech 2-4 (далі id Tech), головним творцем яких є Джон Кармак. Основою швидкодії цих рушіїв є використання цілочисельних обчислень в парі з векторізацією. Прикладом таких ресурсномістких і часто використовуваних завдань є функції: $\frac{1}{x}$, $\frac{1}{\sqrt{x}}$, $\sin x$, тощо. Id Tech використовувався в комерційному ПП Quake III Arena і зарекомендував себе з найкращого боку, вивівши її (гру) до лідерів ігрової індустрії шутерів.

Коротким висновком до даного огляду, присвяченого різного роду принципам та підходам до оптимізації ПП, є наступне ствердження: цілочисельні SIMD обчислення, є широко поширеним підходом, як для збільшення продуктивність ПП, так і для оптимального використання апаратних ресурсів CPU. Далі під продуктивністю/оптимальністю будемо розуміти оптимізацію за допомогою SIMD обчислень, що покриває найбільш ресурсномісткі частини коду ПП.

Виклад основного матеріалу

Тож з урахуванням вищезгаданого огляду програмно-апаратних засобів, треба зробити огляд та внести зауваження щодо виконання СО для майбутнього методу/алгоритму.

Алгоритмічна оптимізація СО, складається з таких пунктів, а саме: умови для використання цілих 32/16-бітних обчислень та квантування даних ядра згортки (1). Почнемо з опису умов для використання цілих 32/16-бітних обчислень, що нададуть змогу для прискорення розрахунку СО ЦЗ. Основою багатьох оптимізованих рішень/обчислень (як показано вище), є цілочисельні операції. Отже, необхідно щоб елементи ядра згортки (1) задовольняли наступним умовам:

$$\Gamma_{i,j} = \nu\gamma_{i,j}, \quad \nu \in R, \quad \gamma_{i,j} \in Z, \tag{2}$$

де ν — коефіцієнт нормалізації.

Надалі, буде описана найбільш ресурсномістка частина обчислень, яка є необхідною для використання у подальших дослідженнях SIMD оптимізації СО (та експериментах пов'язаних з цим).

Будь яке ядро СО можна звести/переформувати до вигляду (2), але найбільш точний результат буде отриманий якщо коефіцієнти $\gamma_{i,j}$ будуть підібрані оптимально, що пов'язано з архітектурними обмеженнями CPU і з SIMD командами мови асемблера. Таким чином, слід вказати/встановити обмеження для коефіцієнтів, щоб уникнути переповнення.

У статті [1, с. 78] було експериментально доведено, що за 16-бітних операціях СО ЦЗ, необхідно враховувати умови (3, 4). Допустимо що кожен піксель зображення, це 8-бітні значення $\in[0\ldots255]$. Те ж стосується і коефіцієнтів/елементів ядра згортки $\gamma_{i,j}$ (або γ). Результат обчислень СО має бути представлений як 16-бітне значення і якщо матриця $\gamma_{i,j}$ не містила негативних елементів, то результат обчислення СО буде описано наступною нерівністю:

$$(2^8-1)\times\sum_{i=0}^{r}\sum_{j=0}^{r}\gamma_{i,j} \le 2^{16}-1. \tag{3}$$

Це означає, що навіть за найбільших результуючих значень зображення та $\gamma_{i,j}$, результат СО не призведе до переповнення та втрати ключових значень. Якщо матриця $\gamma_{i,j}$ містить негативні елементи, то обчислення результату СО буде більш комплексним на етапі квантизації елементів $\gamma_{i,j}$. Отже, будемо використовувати наступу нерівність:

$$(2^8-1)\times\sum_{i=0}^{r}\sum_{j=0}^{r}|\gamma_{i,j}| \le 2^{16-1}-1. \tag{4}$$

І в завершенні опису граничних умов для СО можна коротко резюмувати те, що (3, 4) не мають суттєво вплинути на показники продуктивності/пришвидшення СО у подальших дослідженнях, тому що суттєво, з позиції обчислювальної складності є еквівалентними. Як було показано у [1, с. 78], так і в цьому розділі, пропонується вибрати для даної Γ найбільший ν, такий що $\gamma_{i,j}$ задовольняли умовам (3, 4). Звичайно, варто розглянути/використати і існуючі ядра, як різних форм, так і за змістом. Тож в цій статті, зупинимося на такій коштовній операції як *субполосна* фільтрація (низькочастотна або високочастотна) і т. п. Для цього пропонується використати наступні фільтри (наведені у табл. 2), що задовольняють умовам (3, 4).

Таблиця 2

Квантезовані 8-бітні фільтри для покращення якостей ЦЗ

Призначення	Ядро
Субполосна низькочастотна фільтрація	$\gamma = \begin{pmatrix} 1 & 6 & 1 \\ 6 & 36 & 6 \\ 1 & 6 & 1 \end{pmatrix}, \quad \nu = \frac{1}{64}$
Субполосна високочастотна фільтрація	$\gamma = \begin{pmatrix} -1 & -6 & -1 \\ -6 & 28 & -6 \\ -1 & -6 & -1 \end{pmatrix}, \quad \nu = \frac{1}{64}$
Стабілізація та підвищення різкості	$\gamma = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ -1 & -8 & 74 & -8 & -1 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}, \quad \nu = \frac{1}{42}$
Значна стабілізація та підвищення різкості	$\gamma = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 \\ -1 & -4 & 31 & -4 & -1 \\ 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}, \quad \nu = \frac{1}{15}$
Підвищення контрастності	$\gamma = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 3 & -13 & 3 & 0 \\ 2 & -13 & 48 & -13 & 2 \\ 0 & 3 & -13 & 3 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{pmatrix}, \quad \nu = \frac{1}{16}$

Висновки

У завершенні можна зауважити таке: Для оптимізації вихідного ПП, слід використати SIMD оптимізацію надану, як бібліотеками/фреймворками/ПАПО, так і за допомогою автоматичної векторізації компілятора. У деяких випадках, автоматична векторизація компілятора не надає необхідного/достатнього рівня оптимізації, і в цьому випадку застосовується експертна векторизація, із застосуванням SIMD оптимізованого Inline Assembly коду. Так само, оптимальне використання програмного API ПАПО, для різної архітектури (CPU/GPU і DSP), надасть вихідному ПП ста-більність і високу продуктивність. Умови (3, 4) призводять до необхідності вирішення задачі квантування коефіцієнтів $\Gamma_{i,j}$ і отримання нового «квантизованого» ядра $\gamma_{i,j}$, що у свою чергу, дозволить істотно розширити діапазон застосування 16-біт векторізованих обчислень, як для СО так і в цілому для задачі ЦОС.

Напрямком подальших досліджень є реалізація операції згортки та проведення експериментальних досліджень продуктивності даної

реалізації порівняно з альтернативними функціями, за різних архітектур, при реалізації операції згортки в задачах обробки ЦЗ: ресайзу, фільтрації, потокової фільтрації тощо.

ЛІТЕРАТУРА

1. **Приставка, П. О.**, **Шевченко, А. К**. Дослідження реалізації лінійного оператора згортки цифрового зображення при 16-бітних обчисленнях / Актуальні проблеми автоматизації та інформаційних технологій : зб. наук. праць. – Д. : Вид-во Дніпр. ун-ту., 2016. — Т. 20. — С. 78–90.

2. Documentation for open-cv. (технічний довідник по функціям бібліотеки open-cv) [Електронний ресурс]: 2013. — С 1. — Режим доступу: https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#ga27c049795ce870216ddfb366086b5a04 . – Назва з екрану

3. **Flynn M. J.** Very high speed computers / Michael J. Flynn. // Proceedings of the IEEE: [Vol: 54, Issue: 12], 1966. — P. 1901–1909.

4. ARM® Cortex®—A53 MPCore Processor: (технічний довідник з архітектури процесорів серії Cortex-A53 фірми ARM) [Електронний ресурс]: 2013–2014. — С 620. — Режим доступу: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500g/DDI0500G_cortex_a53_trm.pdf. – Назва з екрану.

5. **Qualcomm** extends hexagon dsp. (технічний довід. з ДСП hexagon v5 фірми Qualcomm) [Електронний документ]: 2013. — С 1. — Режим доступу: pages.cs.wisc.edu/~danav/pubs/qcom/hexagon_microreport2013_v5.pdf. – Назва з екрану.

6. Qualcomm hexagon dsp: An architecture optimized for mobile multimedia and communications. . (презентаційний матеріал щодо можливостей ДСП hexagon фірми Qualcomm) [Електронний ресурс]: 2013. — С 1. — Режим доступу: https://developer.qualcomm.com/download/hexagon/hexagon-dsp-architecture.pdf. – Назва з екрану.

7. **Гриффитс Артур**. GCC. Настольная книга пользователей, программистов и системных администраторов / Артур Гриффитс; [пер. з англ. на рос. ООО «ТИД «ДС» 2004]. — К. : ТИД, 2004. — 624 с.

8. Cardoso Bruno Lopes. Getting Started with LLVM Core Libraries / Bruno Cardoso Lopes, Rafael Auler; — К. : Packt Publishing, 2014. — 314 с.

Шевченко А. К.

ВИБІР ПРОГРАМНО-АПАРАТНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ОПЕРАЦІЇ ЗГОРТКИ З ПІДВИЩЕНОЮ ШВИДКІСТЮ

У статті проведено дослідження програмно-апаратних засобів, для використання в 16-бітних обчислень SIMD технології, які можуть бути застосовані для прискорення операції згортки складової растру цифрового зображення з квадратної маскою лінійного двовимірного фільтра. Базовим, показовим прикладом, для операції згортки слугує функція cv::filter2D(...) бібліотеки OpenCV. Окрім того, виконано критичний аналіз програмно-апаратних засобів для реалізації операцій згортки та для загального підвищення швидкості обробки ЦЗ. Надані висновки та огляд різних апаратних архітектур та умов/засобів, що дозволять пришвидшити обробку медіа даних. Також представлені умови, дотримання яких у перспективі, забезпечить максимальний приріст продуктивності. Також представлені маски фільтра, які були квантезовані в діапазон 8-bit.

Ключові слова: SIMD; ARM NEON; OpenCV; convolution operation.

Shevchenko A. K.

SELECTIONS OF THE SOFTWARE AND HARDWARE APPROACHES AND TECHNIQUES FOR TO IMPROVE SPEED OF CONVOLUTION OPERATION

Base of this article is research of hardware software technic which can achieve significant increase in performance compared to the similar OpenCV functions. A fundamental example of convolution operation implemented is cv::filter2D(...) function of OpenCV library. Critical analysis of software and hardware for implementation of convolution operations and for general speed improvement of image processing were obtained. Conclusions and base review of hardware architectures and conditions/tools that achieved speed up for media processing are presented. Presented conditions, if met thoroughly, ensure maximal increase in performance. Beside, collection of filters quantized to 8-bit range is presented.

Keywords: SIMD; ARM NEON; OpenCV; convolution operation.

Шевченко А. К.

ВЫБОР ПРОГРАММНО-АППАРАТНЫХ СРЕДСТВ ДЛЯ РЕАЛИЗАЦИИ ОПЕРАЦИИ СВЕРТКИ С ПОВЫШЕННОЙ СКОРОСТЬЮ

В статье проведено исследование программно-аппаратных средств, для использования в 16-битных вычислений SIMD технологии, которые могут быть применены для ускорения операции свертки составляющей растра цифрового изображения с квадратной маской линейного двумерного фильтра. Базовым, показательным примером, для операции свертки служит функция cv::filter2D(...) библиотеки OpenCV. Кроме того, выполнен критический анализ программно-аппаратных средств для реализации операций свертки и для общего повышения скорости обработки ЦЗ. Предоставленные выводы и обзор различных аппаратных архитектур и условий/средств, которые позволят ускорить обработку мультимедиа вцелом. Также представлены условия, соблюдение которых в перспективе, обеспечит максимальный прирост по производительности. Также представлены маски фильтра, которые были квантезированы в диапазон 8-bit.

Ключевые слова: SIMD; ARM NEON; OpenCV; convolution operation.

Стаття надійшла до редакції 28.02.2018 р.
Прийнято до друку 28.02.2018 р.
Рецензент — д-р техн. наук, проф. Приставка П. О.