Оптимизация вычислений

Розглянуто питання застосування шаблонів функцій мови програмування C++ для задачі універсального використання мови програмування OpenCL при оптимізації обчислень за допомогою GPU в грід-мережі та на кластерах, побудованих на базі GPU, що використовують різну архітектуру GPU.

УДК 004.4, 004.9, 004.42, 004.67, 004.915, 004.427

А.М. ЛАВРЕНЮК, С.І. ЛАВРЕНЮК

ОПТИМІЗАЦІЯ ОБЧИСЛЕНЬ ЗА ДОПОМОГОЮ УНІВЕРСАЛЬНОГО ВИКОРИСТАННЯ ПРОГРАМИ OPENCL З «ВЕКТОРНИМИ» ТА «СКАЛЯРНИМИ» GPU

Вступ. Незважаючи на інтенсивний розвиток грід-мереж та хмарних обчислень актуальна проблема потреби потужних ресурсів для наукових задач. Постає потреба у високопродуктивних обчисленнях за зниження затрат, адже побудова сучасного кластера чи грід-мережі потребує значних витрат.

Різноманітні організації певний час розглядають потенційні переваги використання графічних процесорів (GPU) для комплексного моделювання та аналізу даних. Адже перехід GPU технології з використання для обробки спеціальних ефектів у 3D графіці для ігор та кіно до використання в наукових дослідженнях не такий уже дивний. Як науковці, так і дизайнери комп'ютерної графіки мають обробляти великі обсяги даних, причому дуже швидко.

Графічні процесори вже зарекомендували себе, як обчислювальні ресурси, у таких областях, як медична обробка зображень і аналіз сейсмічних даних [1, 2]. Графічні процесори ефективні при виконанні швидких розрахунків завдячуючи їх будові, що дозволяє виконувати більше 320 обчислень за одиницю часу на одному GPU.

На основі GPU вже побудовано кластери, які стають також доступні в грід-мережі. Це вимагає універсального підходу до створення програм для роботи з різними GPU.

Векторні операції та GPU. GPU складається з уніфікованих процесорів, які NVIDIA називає уніфікованими потоковими процесорами та являють собою скалярні

процесори загального призначення для обробки даних з плаваючою комою. Традиційно в процесорах існує два типи математики: векторна та скалярна. У випадку векторної математики дані (операнди) представляються у вигляді *n*-мірних векторів, при цьому над великим масивом даних виконується всього одна операція. Найпростіший приклад – колір пікселя задається у вигляді 4-вимірного вектора з координатами R, G, B, A, де перші три координати (R, G, B) задають колір пікселя, а остання – його прозорість. Як простий приклад векторної операції можна розглянути складні кольори двох пікселів. При цьому одна операція виконується одночасно над усіма вісьмома операндами (двома 4-мірними векторами). В скалярній математиці операції виконуються над парою чисел. Зрозуміло, що векторна обробка збільшує швидкість та ефективність обробки за рахунок того, що обробка цілого набору (вектора) даних виконується однією командою.

У графічних процесорах NVIDIA попереднього покоління, а також у графічних процесорах ATI використовується векторна архітектура обчислювальних блоків. Наприклад, у результаті векторні обчислювальні блоки в останніх версіях графічних процесорів ATI можуть виконувати одну векторну операцію за такт для 4-мірних векторів типу float або одну векторну операцію для трьохелементних векторів плюс одну скалярну операцію (схема «3+1»). Векторні обчислювальні блоки в графічних процесорах NVIDIA попередніх поколінь, наприклад, GeForce 6x и GeForce 7x працюють за схемою «2+2».

Останнім часом спостерігається перехід від векторних до скалярних обчислень. При цьому векторний програмний код перетворюється на скалярні операції безпосередньо графічним процесором, наприклад NVIDIA GeForce 8800 [3].

Стандарт OpenCL

Враховуючи, що сучасні комп'ютери часто включають високо паралельні CPU, GPU та інші типи процесорів, важливо дати можливість розробникам програмного забезпечення (ПЗ) використовувати ці гетерогенні процесорні системи в повній мірі.

Створення ПЗ для гетерогенних паралельних обчислювальних платформ є складним процесом, оскільки класичні підходи до програмування для багатоядерних CPU та GPU значно різняться. Хоча в більшості випадків моделі програмування CPU (основані) грунтуються на стандартах, але зазвичай вони пропонують наявність спільного адресного простору та не враховують можливості векторних операцій.

Моделі програмування GPU загального призначення, що включають складні ієрархії пам'яті та векторні операції, традиційно залишаються залежними від платформи. Ці обмеження затрудняють доступ розробників до загальної бази вихідних кодів для CPU, GPU та інших типів процесорів. Як ніколи, нині потрібно надавати можливість розробникам програмного забезпечення ефективно використовувати всі переваги гетерогенних обчислювальних платформ – від високопродуктивних обчислювальних серверів, через персональні комп'ютери, до мобільних пристроїв, які містять різнорідні паралельні CPU, GPU та інші процесори.

OpenCL (Open Computing Language) [4] – відкритий, не потребує ліцензійних відрахувань стандарт для універсального паралельного програмування різних типів процесорів. Стандарт надає програмістам переносний та ефективний доступ до всієї потужності гетерогенних обчислювальних платформ (рис. 1).

У результаті, OpenCL спроможний сформувати базовий рівень паралельного обчислювального коду незалежних від апаратної платформи програмних інструментів, проміжного ПЗ та інших видів програм.

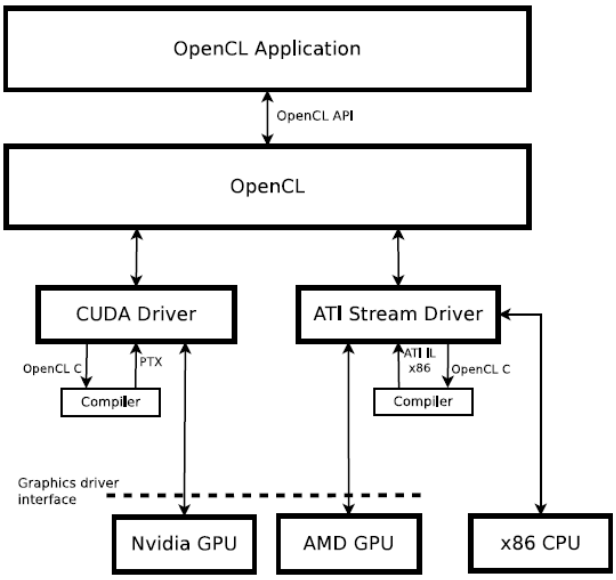

РИС. 1. Модель взаємодії програми на OpenCL з GPU

Проблема універсального використання мови програмування OpenCl на «векторних» та «скалярних» GPU

Але не все так просто при роботі в гетерогенному середовищі, наприклад грід-мережа [5 – 7]. При запуску задач на різних вузлах задача може виконуватися на GPU різної архітектури, з підтримкою чи не підтримкою векторних операцій та певних типів даних програма на OpenCL може поводитись по різному на різних GPU. Це підтверджується проведеними експериментами.

На рис. 2 показано, що при запуску програми на OpenCL та CPU в режимі GPU ми маємо зменшення часу виконання програми при збільшенні вектора даних. Причому це справедливо для трьох основних типів даних (слід зазначити, що тип double підтримується не на всіх GPU, тому з цим типом даних експериментів не проводили).

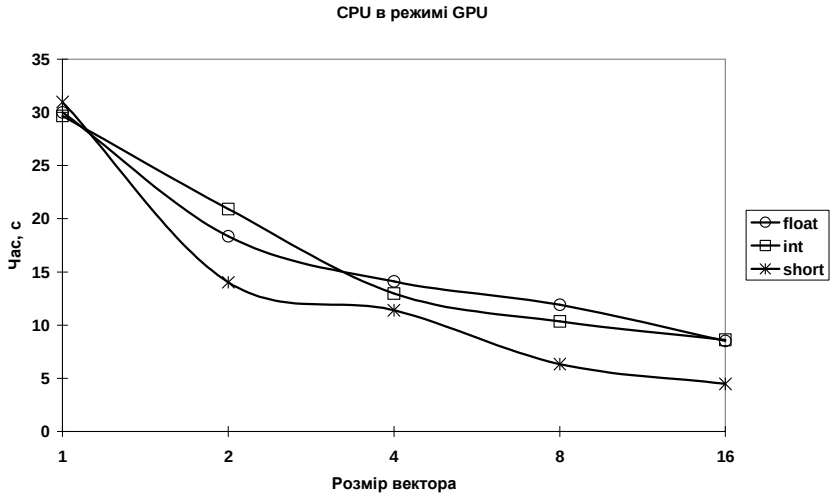

РИС. 2. Залежність часу обчислень від типу даних та довжини вектора даних на CPU в режимі GPU

CPU, що використовувалося в даному експерименті підтримує такі типи даних і довжини векторів: CHAR 16, SHORT 8, INT 4, LONG 2, FLOAT 4, DOUBLE 0.

З рис. 3 видно, що найкращий результат роботи програми не на будь-якій довжині вектора, а тільки на тих довжинах, що підтримують GPU. GPU, що використовувалося в даному експерименті підтримує такі типи даних і довжини векторів: char 16, short 8, int 4, long 2, float 4, double 0.

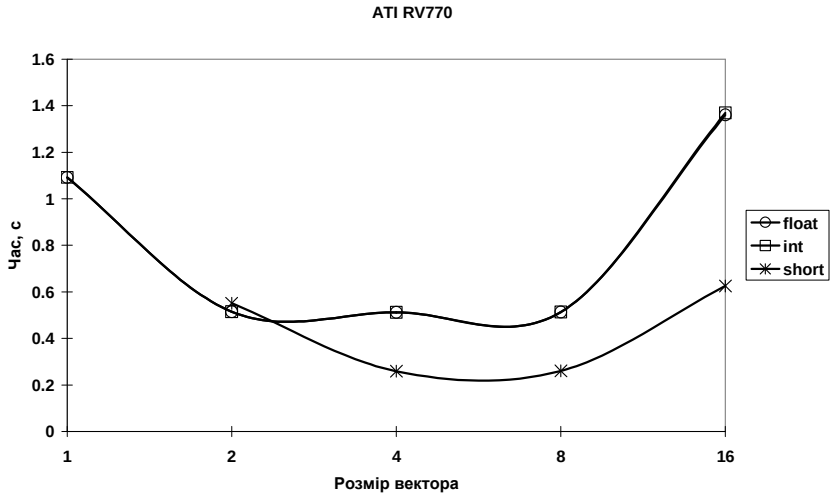

РИС. 3. Залежність часу обчислень від типу даних та довжини вектора даних на графічній карті ATI RV770

GPU, що використано в експериментах (рис. 4 та 5), підтримують такі типи даних і довжини векторів: char 1, short 1, int 1, long 1, float 1, double 1.

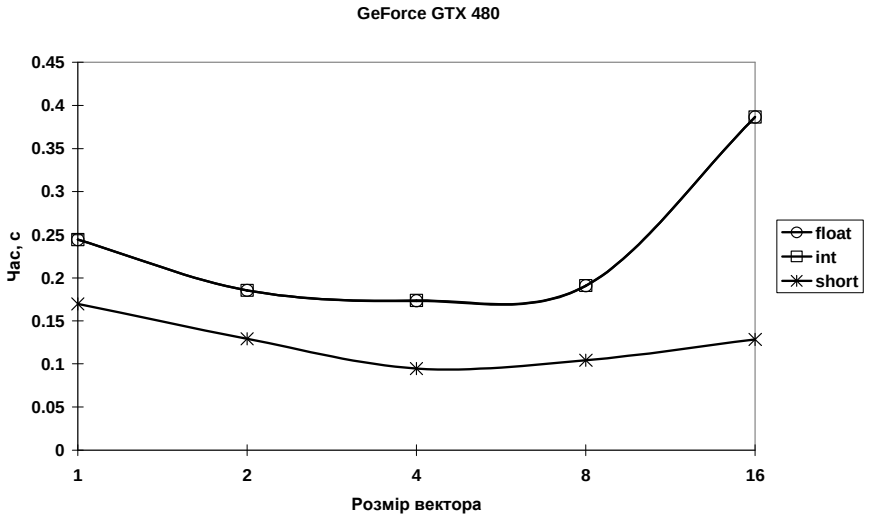

РИС. 4. Залежність часу обчислень від типу даних та довжини вектора даних на графічній карті NVidia GeForce 480

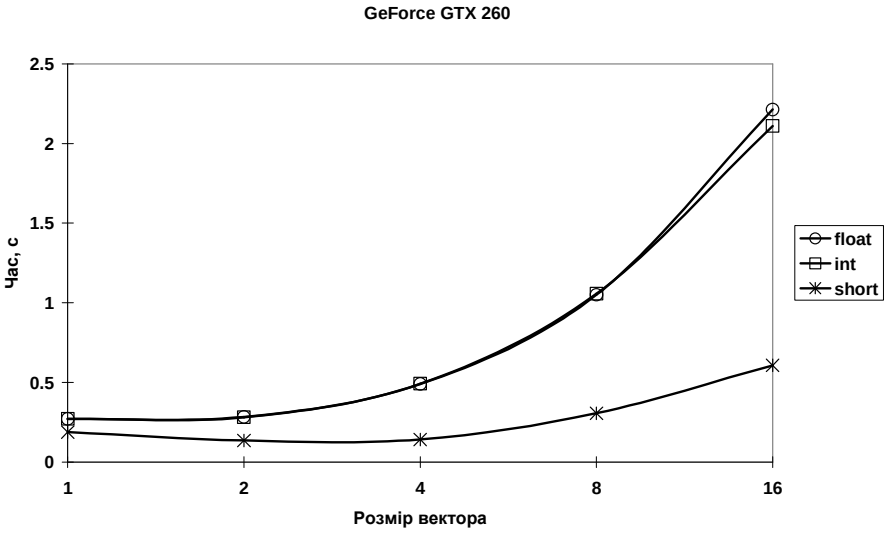

РИС. 5. Залежність часу обчислень від типу даних та довжини вектора даних на графічній карті NVidia GeForce 260

Тобто, всі вектори розбиваються на скаляри і далі над ними виконуються операції. І тут виникає проблема. Програма на OpenCL оптимізована без підтримки векторних операцій може дати не оптимальні обчислення на GPU, що підтримують векторні операції. Програма на OpenCL, яка оптимізована на підтримку векторних операцій, то на одних GPU (рис. 4) при певних довжинах векторів ми можемо не втратити в продуктивності, а на інших (рис. 5) можемо отримати негативний ефект – неоптимальні обчислення, зменшення продуктивності GPU.

Розроблювати декілька різних програм-ядер на OpenCL можна ефективно лише тоді, якщо програма доволі проста. При великих програмах це дуже трудомістка задача і підтримувати різні варіанти програми доволі складно.

Використання шаблонів функцій C++ для універсального використання мови програмування OpenCl на «векторних» та «скалярних» GPU.

Для універсалізації програмування OpenCl на «векторних» та «скалярних» GPU запропоновано наступний підхід, блок-схему якого показано на рис. 6.

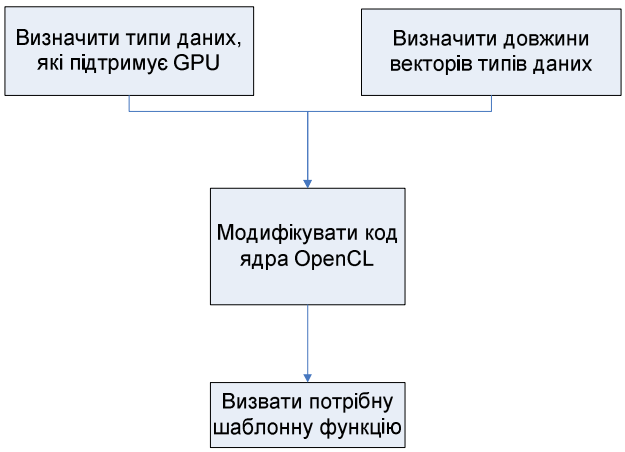

РИС. 6. Універсалізація програми OpenCL

Програмно легко визначити типи даних та довжини векторів, що підтримує GPU. Так як програма-ядро на OpenCL зберігається у вигляді текстової змінної типу char *, то не важко в ядрі модифікувати рядок програми (на жаль поки OpenCL не підтримує шаблони, так як ця мова є похідною від C, а шаблони функцій введені в мові C++ [8]):

```
typedef __global int4 mtype.
```

А ядро програми на OpenCL залишати не змінним. І в ньому використовувати перевизначений вище тип даних:

```
__kernel void kernel_test(  mtype * A, mtype * B, mtype * C, mtype * D,
const unsigned int nx_max) {
   int j=get_global_id(1);
   int i=get_global_id(0);
   int nx_BLOCK=nx_max;
   A[j*nx_BLOCK+i]=(B[j*nx_BLOCK+i]-
   C[j*nx_BLOCK+i])*D[j*nx_BLOCK+i];}
```

З програмою, що буде виконуватися на хості та підготовлювати дані для ядра і запускати на виконання програму-ядро на OpenCL, ситуація трохи складніша. Динамічно перевизначити тип даних складно.

Як вихід запропоновано та перевірено наступний підхід.

Зробити мінімум дві функції, перша визначає типи даних, довжини векторів, та викликає на виконання функцію, що визначена на основі шаблона.

Приклад такої функції, що викликає шаблонну функцію:

```
main_run()
{
   ret=gettype(type1, len1); //за основу gentype можна взяти відомий
   приклад програми Nvidia DeviceQuery
   if(len1==4){
      const int N=4;
      if(type1==1) {
         float f[N];
         ret=run(f); }
      if(type1==2) {
         int t[N];
         ret=run(t); }
      if(type1==2) {
         short s[N];
         ret=run(s); }
      }
   …
   }
```

Приклад шаблонної функції для підготовки даних для програми-ядра на OpenCL та запуску програми-ядра на GPU:

```
template <class Type, int size> int run( const Type (&r_array)[size] )
{
   const int loc_size = size;
   typedef union {Type s[loc_size];} mtype;
//Створюємо векторний тип на основані типу Type,
  довжина вектора loc_size
   mtype *B=NULL;
   int sizeXZ=nx_max*ny_max;
   int nx_max4=nx_max/loc_size;
```

```
int sizeXZ4=nx_max4*ny_max;
B=new mtype[sizeXZ4];
for(int j=1;j<ny_max-1; j++)
   for(int i=1;i<nx_max-1; i++)
      for(int k=0;k<loc_size;k++)
         B[j*nx_max+i].s[k]=(Type) 123.456f*i;
// Тут вирішуємо проблему різної довжини векторів
// Та вирішуємо проблему приведення до векторного
   типу mtype через приведення до базового типу (Type).
```

На перший погляд здається, що все це можна було б вирішити за допомогою векторних типів даних, таких як cl_float4, cl_int8. Але тут виникає проблема при довжині вектора 1. Так, наприклад, при оголошенні cl_float *B і виконанні коду B[j*nx_max+i].s[k]=… буде виникати помилка. А вставка операцій умовного оператора, для того, щоб обійти помилку, буде уповільнювати роботу програми (при оптимізації програми на C++ рекомендовано уникати використання else в операторі if) та зменшувати зручність роботи з кодом:

```
for(int j=1;j<ny_max-1; j++)
   for(int i=1;i<nx_max-1; i++)
   if(loc_size==1)
      B[j*nx_max+i]=(Type) 123.456f*i;
   else
      for(int k=0;k<loc_size;k++)
         B[j*nx_max+i].s[k]=(Type) 123.456f*i.
```

При запуску програми-ядра на OpenCL варто врахувати те, що розмір рядків матриці буде зменшено, пропорційно до довжини вектора:

```
nx_max4=nx_max/loc_size.
```

І функцію, що запускає на виконання програму-ядро на OpenCL слід викликати наступним чином:

```
size_t globalgroup[3]={nx_max4, ny_max, 1};
err=clEnqueueNDRangeKernel ( queue, kernel, work_dim, NULL,
globalgroup,  NULL , 0, NULL, NULL).
```

Висновки. В роботі наведено результати використання одного з підходів до підвищення продуктивності програм на мові OpenCL при запуску їх на GPU пристроях різної архітектури.

Запропонований підхід дає можливість уніфікувати програму для роботи з різними GPU пристроями з малими затратами, прозорістю програмного коду та легкістю його подальшої підтримки та модернізації; використовувати переваги векторних операцій на GPU, які підтримують векторні операції; уникати проблем, які можуть виникати, при запуску «векторної» програми на GPU пристроях, що не підтримують векторні операції.

А.Н. Лавренюк, С.И. Лавренюк

ОПТИМИЗАЦИЯ ВЫЧИСЛЕНИЙ С ПОМОЩЬЮ УНИВЕРСАЛЬНОГО ИСПОЛЬЗОВАНИЯ ПРОГРАММЫ OPENCL С «ВЕКТОРНЫМИ» И «СКАЛЯРНЫМИ» GPU

Рассмотрен вопрос применения шаблонов функций языка программирования C++ для задач универсального использования языка программирования OpenCL при оптимизации вычислений с помощью GPU в грид-сети и на кластерах, построенных на базе GPU, которые используют различную архитектуру GPU.

A.N. Lavreniuk, S.I. Lavreniuk

OPTIMIZATION OF COMPUTATIONS WITH UNIVERSAL USE OF OPENCL PROGRAM WITH «VECTOR» AND «SCALAR» GPU

Using template functions from C ++ programming language for the problems of universal use of OpenCl programming language in optimization of computations with GPU in the grid and on the GPU-based clusters, which use different architecture of GPU, is considered.

1. *LI Bo, LIU Guo-feng, LIU Hong.* A method of accelerating seismic Pre-stack time migration by GPU // [Електронний ресурс] – http://cm.seg.org/documents/10161/997244/1202.pdf.
2. *Virasin Archirapatkave, Hendra Sumilo, Simon Chong Wee See, Tiranee Achalakul.* GPGPU Acceleration Algorithm for Medical Image Reconstruction // ISPA '11 Proceedings of the 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications table of contents P. 41–46 Publisher IEEE Computer Society Washington, USA.
3. *Графические* процессоры GeForce [Електронний ресурс] http://www.nvidia.ru/object/geforce_family_ru.html
4. *OpenCL* – The open standard for parallel programming of heterogeneous systems [Електронний ресурс] http://www.khronos.org/opencl/.
5. *Загородна Н.В., Лупенко С.А., Луцків А.М.* Особливості створення grid-систем на основі gpu-вузлів для розв'язання задач криптоаналізу // Інформаційні системи та мережі. – Вісник Національного університету "Львівська політехніка" № 699. – Львів: Видавництво Львівської політехніки, 2011 – С. 312–320.
6. *Chris* Jang OpenCL™ Optimization Case Study: GATLAS - Designing Kernels with Auto-Tuning **//[**електронний ресурс**] –** http://golem5.org/gatlas/CaseStudyGATLAS.htm.
7. *Marcus Hinders.* GPU Computations in Heterogeneous Grid Environments, Joint Research Report **//** [Електронний ресурс] **–**http://www.techila.fi/technology/technology-docs/.
8. ISO/IEC 14882:1998 Programming languages – C++ //[Електронний ресурс] – http://www.iso.org/iso/catalogue_detail.htm?csnumber=25845

Отримано 14.12.2011

Про авторів:

Лавренюк Алла Миколаївна,
кандидат технічних наук, доцент Національного технічного університету України «КПІ»,

Лавренюк Сергій Іванович,
молодший науковий співробітник відділу автоматизації програмування
Інституту кібернетики імені В.М. Глушкова НАН України.