УДК 004.4'24

А.Ю. Дорошенко, О.А. Яценко, О.Г. Бекетов

АЛГОРИТМ АВТОМАТИЗОВАНОГО РОЗПАРАЛЕЛЮВАННЯ ЦИКЛІЧНИХ ОПЕРАТОРІВ ДЛЯ ГРАФІЧНИХ ПРИСКОРЮВАЧІВ

Розроблено програмний засіб для оптимізації обчислень, що дозволяє в напівавтоматичному режимі здійснити паралелізацію циклічних операторів програми для виконання обчислень на графічних прискорювачах. Здійснено буферизацію даних, синхронізовану із виконанням основного циклу, та побудований за допомогою системи переписувальних правил TermWare засіб інтегровано з інструментарієм проектування та синтезу програм (ІПС). Проведено випробування розробленої системи на гетерогенному мультиядерному кластері.

Ключові слова: методи паралелізації, оптимізація циклів, обчислення загального призначення на графічних процесорах, проектування та синтез програм.

Вступ

Розпаралелювання циклічних операторів є давно відомою проблемою паралельного програмування. З широким використанням графічних прискорювачів для обчислювальних задач виникла нова постановка цієї проблеми для цього класу мультиядерних систем. Мета даної статті – вдосконалення механізму перетворень операторів циклу для його паралелізації для виконання на графічному прискорювачі, наведеного в попередній статті [1]. Недоліком попередньої схеми перетворень було використання попередньої лінеаризації даних перед їх буферизацією, що суттєво ускладнювало подальшу процедуру передачі даних, оскільки потребувало окремого механізму контролю синхронізації. В даній роботі натомість реалізовано буферизацію даних, синхронізовану із виконанням основного циклу. Крім того, з метою підтвердження концепції було створено автоматизований інструментарій перетворення циклів за допомогою системи переписувальних правил TermWare і інтегрування його з інструментарієм проектування та синтезу програм ІПС. Наведено порівняння властивостей програм, отриманих при застосуванні наведеного перетворення та за допомогою системи автоматизованої паралелізації Par4All [2].

1. Механізм перетворення циклу

Нехай задане гніздо циклу, складене з циклів із лічильником, таке, що його ітерації незалежні між собою, а області значень ітераторів задані статично.

Графічний прискорювач є багатопотоковим обчислювальним пристроєм з SIMD архітектурою, тобто окремі паралельні потоки виконують один спільний набір команд, але над різними даними. Оскільки ітерації циклу незалежні, можливе їх виконання паралельними потоками. Отже, слід розбити цикл на окремі ітерації так, щоб кожна ітерація виконувалась окремим потоком, і кожному потоку надати набір даних, що відповідає виконуваній ітерації.

Наявність власної оперативної пам’яті призводить до необхідності обміну даними з центральним пристроєм. Обсяг даних, що обробляються, може перевищувати обсяг пам’яті графічного прискорювача, тому дані слід передавати порціями. У тому випадку, коли дані, що обробляються, не вміщуються у пам’ять графічного прискорювача, або у випадку обробки неперервного потоку даних, постає необхідність розбиття початкового циклу на окремі підцикли.

Розглянемо складений цикл, що має наступний вигляд:

$$\begin{array}{l} for \quad i_0 = 0...\#I_0 \\ \quad for \quad i_1 = 0...\#I_1 \\ \quad\quad ... \\ \quad\quad for \quad i_N = 0...\#I_N \\ \quad\quad\quad F(\bar{i}, \overline{p^{in}(\bar{i})}, \overline{p^{out}(\bar{i})}); \end{array} \qquad (1)$$

де $I_0, I_1, ..., I_N$ – множини значень індексів $i_0, i_1, ..., i_N$; $N+1$ – глибина вкладеності циклу, символом $\bar{i}$ позначено множину $\bar{i} = \{ i_j \mid j = 0, ..., N \}$;

$\overline{p^*(\bar{i})} = P^* = \{ p_j^*(\bar{i}) \mid j = 0...\#P^* \}, * \in \{in, out\}$ – множини значень параметрів вхідних та вихідних даних, F – відображення, що здійснює перетворення даних. Для спрощення припустимо, що множини вхідних та вихідних параметрів не перетинаються.

Позначимо T кількість потоків, що будуть задіяні при виконанні ядра *kernel*. Після перетворень цикл набуде наступного вигляду:

$$
\begin{aligned}
&for\ \ e = 0 : L \\
&\quad fill(inBuf); \\
&\quad push(inBuf); \\
&\quad kernel(e, inBuf, outBuf); \\
&\quad pull(outBuf); \\
&\quad unpack(outBuf);
\end{aligned}
\qquad (2)
$$

де L – кількість викликів ядра, що обирається таким чином, щоб $L \cdot T$ не перевищувало G – загальної кількості ітерацій початкового циклу; *fill* – функція заповнення буферу початкових даних; *push* – переміщення буферу початкових даних у пам'ять прискорювача; *kernel* – виклик ядра; *pull* – переміщення оброблених даних із пам'яті прискорювача до буферу оброблених даних; *unpack* – копіювання даних із буферу оброблених даних у відповідні змінні. Функція *fill* має наступний вигляд:

$$
\begin{aligned}
&for\ \ t = 0 : T \\
&\quad id = e \cdot T + t; \\
&\quad inbuf_t = p^{in}(g(id)).
\end{aligned}
$$

Таким чином, *id* визначає номер поточної ітерації. Тут g – функція, що співставляє номеру ітерації відповідний набір значень ітераторів циклу [1], $inbuf_t$ – буфер вхідних даних ітерації *t*. Функція *kernel* складається із виклику вмісту початкового циклу:

$$F(\bar{i}, \overline{inbuf_{id}(\bar{i})}, \overline{outbuf_{id}(\bar{i})}),$$

де *id* – номер потоку графічного прискорювача, що виконує ітерацію. Функція *unpack* аналогічна *fill*:

$$
\begin{aligned}
&for\ \ t = 0 : T \\
&\quad id = e \cdot T + t; \\
&\quad p^{out}(g(id)) = outbuf_t .
\end{aligned}
$$

Таким чином, при паралелізації окремого циклу структура основної програми залишається незмінною.

На рис. 1 наведено діаграму послідовності паралельної програми для одного прискорювача. Обчислення виконуються у три потоки, що одночасно виконують функції *fill*, *kernel* та *unpack*.

2. Реалізація методу

За допомогою системи символьних обчислень TermWare [3] було реалізовано експериментальний інструмент LoopRipper, що дозволяє, маючи списки вхідних та вихідних змінних, згенерувати функції *kernel*, *fill* та *unpack*, із яких паралельна програма компонується за допомогою інструментарію проектування та синтезу програм [4]. Генерація функцій здійснюється шляхом заміни параметрів вхідним та вихідним буферами даних та перерозрахунком ітераторів у заданому початковому циклі. Таким чином, від користувача вимагається надати цикл та списки параметрів. Подальша автоматизація процесу потребує аналізу дерева залежностей параметрів.

3. Інтеграція з системою ІПС

З метою автоматизації проектування та перетворення (паралелізації) програми у даній роботі спільно з системою TermWare [3] застосовується інструментарій проектування та синтезу програм (ІПС) [4].

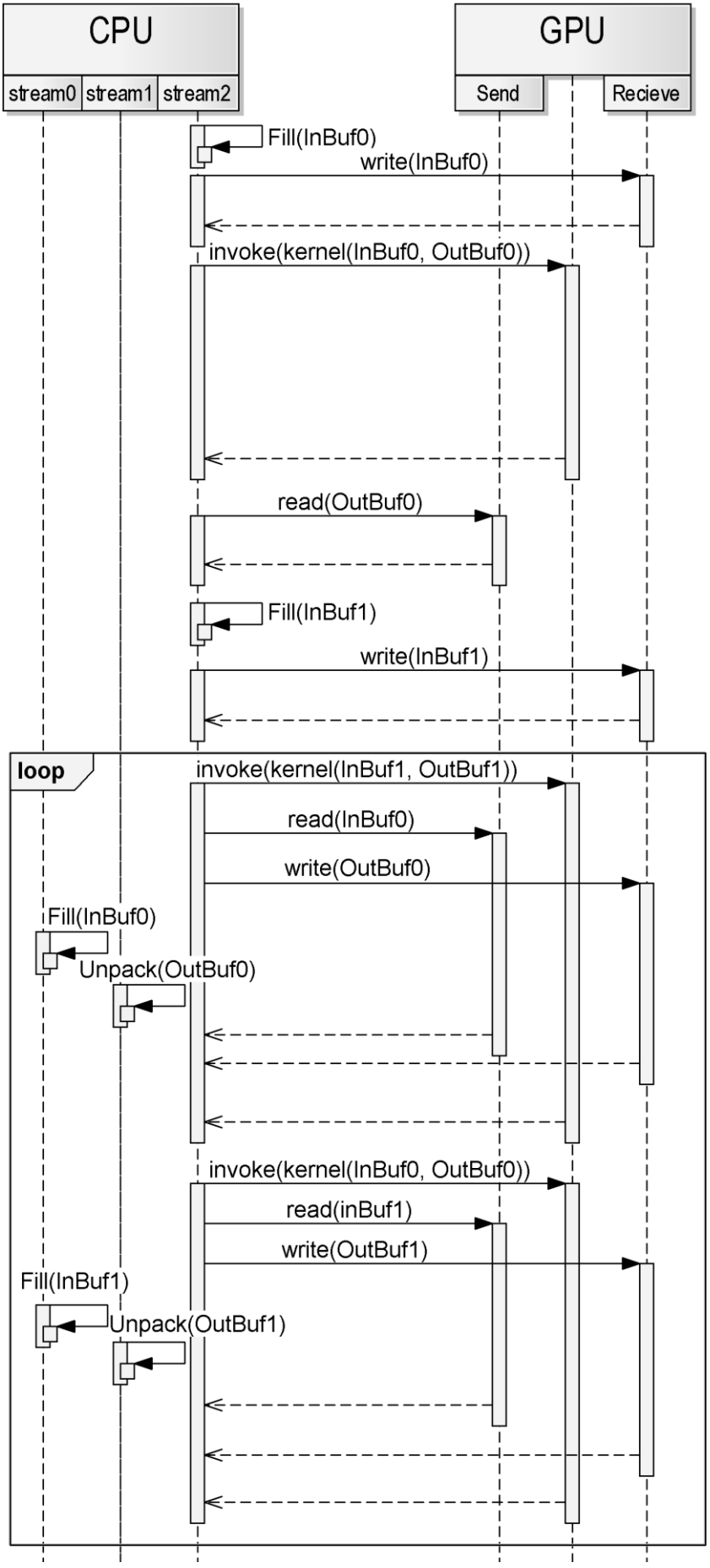

Рис. 1. Діаграма послідовності паралельної програми для одного прискорювача

Система ІПС призначена для конструювання послідовних і паралельних САА-схем алгоритмів, поданих в модифікованих системах алгоритмічних алгебр В.М. Глушкова (САА-М), а також генерації програм мовами С++, Java, C for CUDA та ін. Основним компонентом системи є діалоговий конструктор синтаксично правильних програм (ДСП-конструктор), за допомогою якого виконується порівневе проектування САА-схем зверху вниз шляхом деталізації мовних конструкцій САА-М. На кожному кроці проектування система надає користувачу на вибір лише ті конструкції, підстановка яких у текст схеми, що формується, не порушує її синтаксичну правильність. ДСП-конструктор використовує список конструкцій САА-М і дерево конструювання алгоритму. Специфікації конструкцій та їх відображення в цільові мови програмування зберігаються в базі даних інструментарію.

Основними етапами процесу розробки та трансформації програми (у рамках задачі паралелізації циклів, що розглядається у даній роботі) за допомогою ІПС та TermWare є такі:

1) конструювання САА-схеми послідовного алгоритму за допомогою ІПС та генерація відповідного коду цільовою мовою програмування;

2) трансформація коду послідовного циклу у функцію-ядро та генерація додаткових структур даних та макровизначень за допомогою TerwWare;

3) конструювання САА-схеми шаблону виклику ядра у системі ІПС та заміна послідовного циклу на виклик ядра;

4) вставка коду, згенерованого в TerwWare, у вихідну програму за допомогою системи ІПС;

5) генерація паралельної програми цільовою мовою програмування (C for CUDA) на основі отриманої САА-схеми в ІПС.

Приклад. Запропонований метод застосовано для розпаралелювання послідовного циклу з глибиною гнізда рівною чотирьом, що входить до складу програми чисельного прогнозування погоди, розглянутої в [5], а саме циклу інтерполяції початкових даних у вузлах локальної сітки. Далі наведено фрагмент послідовної САА-схеми, сконструйований в системі ІПС.

```
FOR (h FROM 0 TO Pk - 1)
LOOP
  FOR (k FROM 0 TO Lmz - 1)
  LOOP
    FOR (j FROM 0 TO Mmz - 1)
    LOOP
      FOR (i FROM 0 TO Nmz - 1)
      LOOP
        "(a) := ((WZZ + US[h][k][j][i] /
          0.321) * Rs * VS[h][k][j][i])";
        "(Tp) := (TS[h][k][j][i] *
          pow(1000.0 / HS[h][k][j][i],
           2.0/7.0))";
        "(Tv) := (Tp * (1.0 + 0.6078 *
          QS[h][k][j][i]))";
        "(Qc[h][k][j][i]) := (a - (0.5 * Tv +
          (1.0 - Zmz[k]) * g * F_X[j][i] /
          0.321))"
      END OF LOOP
    END OF LOOP
  END OF LOOP
END OF LOOP;
```

На основі наведеної САА-схеми в ІПС виконана генерація програмного коду мовою С. За допомогою TerwWare здійснене перетворення тіла послідовного циклу у функцію-ядро (Kernel) та генерація додаткових даних і макровизначень. В системі ІПС спроектовані САА-схеми шаблону виклику ядра (складеного оператора GPU) та функції-ядра (складеного оператора Kernel), що наведені далі.

```
"GPU" ====
= "Declare a variable (threadNum) of
    type (int) = (threadNum_def)";
  "Declare a variable (threadsPerBlock) of
    type (int) = (threadsPerBlock_def)";
```

```
"Declare a variable (gIterNum) of
 type (int) = (gIterNum_def)";
"Declare a variable (launchNum) of
 type (int)";
 (launchNum :=
  ceiling(gIterNum, threadNum));
"Declare a variable (blocksPerGrid) of
 type (int)";
(blocksPerGrid :=
 ceiling(threadNum, threadsPerBlock));
"Allocate CPU memory for input and output
 buffers (HI0, HI1, HO0, HO1)";
"Allocate GPU memory for input and
 output buffers (DI0, DI1, DO0, DO1)";
FOR (t FROM 0 TO threadNum - 1)
LOOP
  "Declare a variable (id) of type (int) =
   (0*threadNum + t)";
  "Copy parameters to input data buffer
    (HI0)"
END OF LOOP;
"Copy variable (HI0) from CPU memory to
 variable (DI0) of size (threadNum *
 sizeof(inDataChunk)) on GPU";
FOR (t FROM 0 TO threadNum - 1)
LOOP
  "Declare a variable (id) of type (int) =
   (1*threadNum + t)";
  "Copy parameters to input data buffer
    (HI1)"
END OF LOOP;
FOR (e FROM 0 TO launchNum - 1)
LOOP
  ("Copy data from (HI0, HI1) to (buf)
     according to (e)"

  PARALLEL

  (IF 'Even number (e)'
   THEN
      "Copy variable (HI1) from CPU
       memory to variable (DI1) of size
       (threadNum * sizeof(inDataChunk))
       on GPU in the stream (stream1)
       asynchronously";
      Call_GPU_kernel(blocksPerGrid,
                threadsPerBlock, 0, stream2)
      (
        "Kernel(DI0, DO0, threadNum)"
      );
      "Copy variable (DO1) from GPU
       memory to variable (HO1) of size
       (threadNum*sizeof(outDataChunk))
       on CPU asynchronously in the
       stream (stream3)"
   ELSE
     "Copy variable (HI0) from CPU
      memory to variable (DI0) of size
      (threadNum * sizeof(inDataChunk))
      on GPU in the stream (stream1)
      asynchronously";
     Call_GPU_kernel(blocksPerGrid,
                threadsPerBlock, 0, stream2)
     (
       "Kernel(DI1, DO1, threadNum)"
     );
     "Copy variable (DO0) from GPU
      memory to variable (HO0) of size
      (threadNum*sizeof(outDataChunk))
      on CPU asynchronously in the stream
      (stream3)"
   END IF;
   WAIT 'All threads completed work'

   PARALLEL

  "Copy data from (HO0, HO1) to (outbuf)
    according to (e) and unpack outbuf"
   ))
END OF LOOP;
"Copy data from (HO0, HO1) to (outbuf)
 according to (launchNum-1) and unpack
```

```
   outbuf";
  "Free the memory for variable (DI0) on
   GPU";
  "Free the memory for variable (DO0) on
   GPU";

"Kernel(inDataBuf, outDataBuf, threadNum)"
==== "Insert program code from the file
       (kernel.cu)";
```

У вищенаведеному складеному операторі GPU виклики функції-ядра виконуються асинхронно за допомогою операції

```
Call_GPU_kernel(blocksPerGrid,
        threadsPerBlock, 0, stream)
(
  "Kernel(inDataBuf, outDataBuf,
          threadNum)"
),
```

де blocksPerGrid – кількість блоків у сітці; threadsPerBlock – кількість потоків у кожному блоці; stream – черга виконання функцій-ядер; Kernel – функція-ядро; inDataBuf, outDataBuf – вхідний та вихідний буфери даних, відповідно; threadNum – кількість потоків. (Більш детально операції САА-М, призначені для формалізації обчислень на GPU, розглядаються в роботі [4]).

У складеному операторі Kernel зазначено базисний оператор вставки програмного коду функції-ядра мовою С з файлу, згенерованого за допомогою TerwWare:

```
"Insert program code from the file
 (kernel.cu)"
```

Код згаданої функції-ядра має вигляд:

```
__global__ void Kernel(
    inDataChunk *inDataBuf,
    outDataChunk *outDataBuf,
      int threadNum)
{
  LocalDataInit;

  int myId = blockDim.x*blockIdx.x +
             threadIdx.x;  // cuda grid id

 if (myId < threadNum)
 {
   a = (WZZ + inDataBuf[myId].a0 / 0.321)*
       Rs * inDataBuf[myId].a1;
   Tp = inDataBuf[myId].a2 * pow(1000.0 /
        inDataBuf[myId].a3, 2.0/7.0);
   Tv = Tp * (1.0 + 0.6078 *
        inDataBuf[myId].a4);
   outDataBuf[myId].a0 = a - (0.5 * Tv +
        (1.0 - inDataBuf[myId].a5) * g *
        inDataBuf[myId].a6 / 0.321);
 }
}
```

Вставка згенерованих у TermWare додаткових структур даних та макровизначень у паралельну програму здійснюється також за допомогою базисного оператора вставки коду з файлу.

На основі побудованої паралельної САА-схеми алгоритму в ІПС було виконано генерацію паралельної програми мовою C for CUDA, результати виконання якої на наведено у наступному розділі.

4. Експеримент

Для визначення ефективності перетвореної програми проведено експеримент, що складається із двох частин.

В першій частині здійснено порівняння результатів виконання програм, отриманих за допомогою описаної системи перетворень та системи автоматичної паралелізації Par4All, що використовує поліедральний метод паралелізації циклів і дозволяє генерувати програми для графічних процесорів на рівні коду CUDA в Linux системах. Система Par4All має суттєві обмеження. По-перше, ця система не дозволяє працювати із обсягом даних, що

перевищує обсяг пам'яті відеокарти. По-друге, як виявилося при застосуванні Par4All до експериментальної програми, система не спроможна виконати обробку вказівників, тому довелося відмовитись від їх використання. При проведенні першої частини експерименту використовувалися лише статичні масиви із залученням стеку великих обсягів (понад 2 Гб). Експеримент виконано на операційній системі Ubuntu 16.04 із використанням компілятора nvcc 8.0.61.

В другій частині порівнюються результати виконання перетвореної програми для однієї та для двох відеокарт. Par4All не передбачає можливості використання більш ніж одного прискорювача одночасно, тому в експерименті задіяна лише система LoopRipper.

Експеримент проводився з використанням апаратної системи, оснащеної процесором Intel Core i5-3570 (4 ядра, 3.8 ГГц), оперативною пам'яттю обсягом 8 Гб та відекартами NVIDIA Tesla M2050 (3 Гб оперативної пам'яті, ширина шини передачі даних 384 біт, 448 ядер CUDA, підключення через інтерфейс PCI Express x16) та NVIDIA GeForce GTX 650Ti (1 Гб оперативної пам'яті, ширина шини передачі даних 128 біт, 768 ядер CUDA, підключення через інтерфейс PCI Express x1). Попри більшу кількість ядер, друга відеокарта має повільнішу шину обміну даних, до того ж, підключення через повільний інтерфейс PCIex1 додатково обмежує швидкість передачі, тому розподіл навантаження між відеокартами встановлено на рівні 4:1. При такому розподілі час на виконання обчислень у обох відеокарт приблизно однаковий.

Масштабування дослідної задачі здійснювалось шляхом вибору розміру локальної сітки і варіювалось у межах доступної пам'яті центрального пристрою. Параметр *T* (кількість потоків відеокарти, задіяних при виклику ядра) підібрано таким чином, щоб мінімізувати час виконання програми. Після виконання результати послідовної та паралельної програм порівнювались між собою.

Як видно із графіка першого експерименту (рис. 2), експериментальна система випереджає Par4All, коефіцієнт відносного прискорення досягає 1.45. При виконанні експерименту з двома відеокартами (рис. 3) використовувалось динамічне

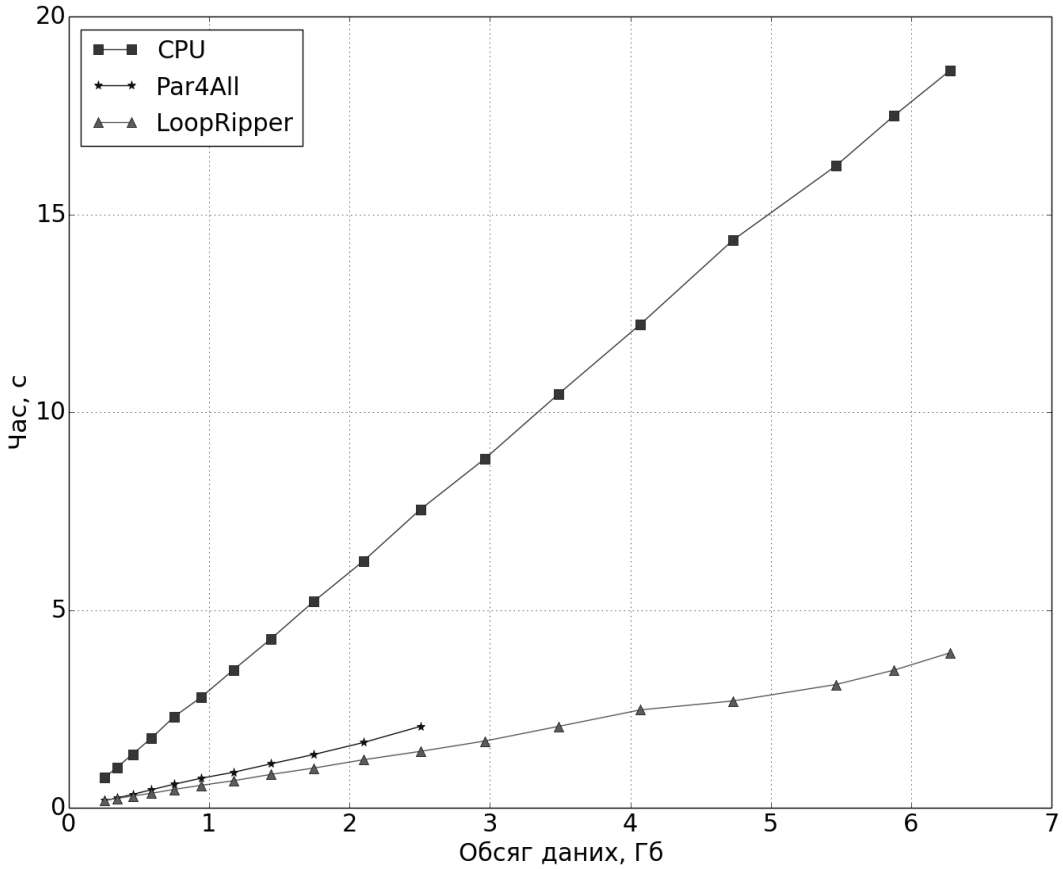

Рис. 2. Графік першого експерименту

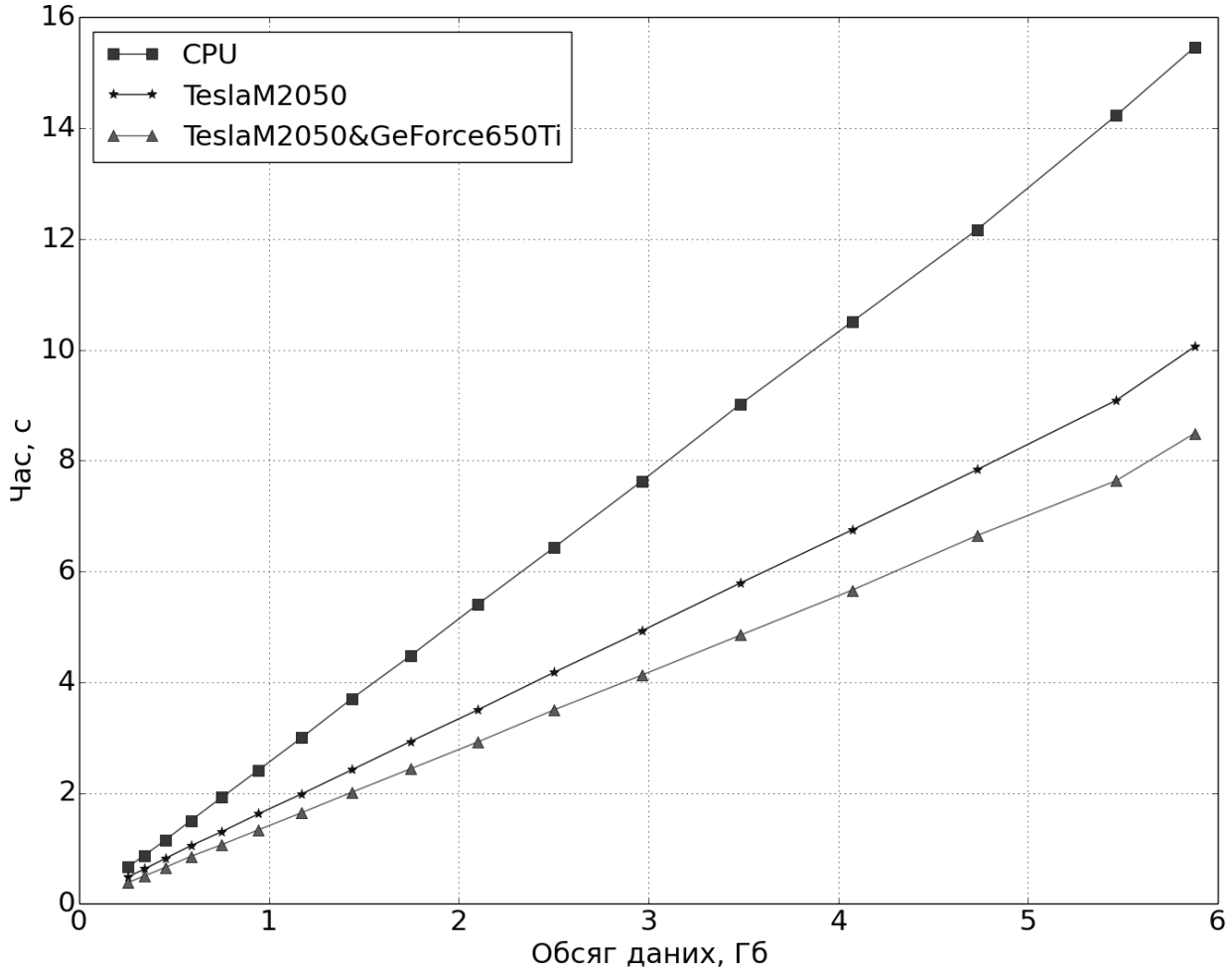

Рис. 3. Графік другого експерименту

керування пам'яттю, експеримент виконано на операційній системі Windows 7 із використанням компілятора nvcc 5.5.0. Оскільки повна паралелізація за схемою із копіюванням даних в окремих потоках потребує трьох потоків на відеокарту, при наявних чотирьох ядрах центрального процесора для уникнення конкуренції між потоками копіювання даних виконувалось послідовно із запуском ядра, тобто із залученням по одному потоку для кожної відеокарти. При залученні додаткової відеокарти зростання швидкості наближається до 1.2 рази, що пропорційно розподілу навантажень. Було виявлено відставання програм, виконуваних в системі Ubuntu, в порівнянні з системою Windows, причини якого наразі не з'ясовано.

Висновки

Реалізовано програмний засіб паралелізації циклів для оптимізації обчислень за допомогою графічних прискорювачів, що дозволяє в напіватоматичному режимі здійснити перехід від послідовної до паралельної програми. Проведено порівняння з відомою автоматизованою системою паралелізації Par4All, виявлено переваги розробленої системи у швидкодії, можливості обробки обсягів даних, що перевищують обсяг пам'яті графічного прискорювача та можливості залучення декількох прискорювачів одночасно.

1. Дорошенко А.Ю., Бекетов О.Г. Метод паралелізації циклів сіткових обчислювальних задач для графічних прискорювачів. *Проблеми програмування.* 2017. № 1. С. 59–66.
2. PIPS: Automatic Parallelizer and Code Transformation Framework [Електронний ресурс]. Режим доступу до ресурсу: http://pips4u.org/

3. Doroshenko A., Shevchenko R. A rewriting framework for rule-based programming dynamic applications. *Fundamenta Informaticae*. 2006. Vol. 72, N 1–3. P. 95–108.
4. Андон Ф.И., Дорошенко А.Е., Жереб К.А., Шевченко Р.С., Яценко Е.А. Методы алгебраического программирования: формальные методы разработки параллельных программ. Киев: Наукова думка, 2017. 440 с.
5. Дорошенко А.Ю., Бекетов О.Г., Прусов В.А., Тирчак Ю.М., Яценко О.А. Формалізоване проектування та генерація паралельної програми чисельного моделювання погоди. *Проблеми програмування*. 2014. № 2–3. С. 72–81.

References

1. Doroshenko A.Yu., Beketov O.G. (2017) Method of parallelization of loops for grid calculation problems on GPU accelerators. Problems in programming. (1). P. 59–66. (in Ukrainian).
2. PIPS: Automatic Parallelizer and Code Transformation Framework [Online]. Available from: http://pips4u.org/
3. Doroshenko A. & Shevchenko R. (2006) A rewriting framework for rule-based programming dynamic applications. Fundamenta Informaticae. 72 (1-3). P. 95–108.
4. Andon P.I. et al. (2017) Methods of algebraic programming: formal methods of parallel program development. Kyiv: Naukova dumka. (in Russian).
5. Doroshenko A.Yu., Beketov O.G., Prusov V.A., Tyrchak Yu.M. & Yatsenko O.A. (2014) Formalized designing and generation of parallel program for numerical weather forecasting task. Problems in programming. (2-3). P. 72–81. (in Ukrainian).

Одержано 10.10.2017

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень
Інституту програмних систем
НАН України,
професор кафедри автоматики
і управління в технічних системах НТУУ
"КПІ імені Ігоря Сікорського".
Кількість наукових публікацій в
українських виданнях – понад 150.
Кількість наукових публікацій в
зарубіжних виданнях – понад 50.
Індекс Хірша – 5.
http://orcid.org/0000-0002-8435-1451,

Яценко Олена Анатоліївна,
кандидат фізико-математичних наук,
старший науковий співробітник
Інституту програмних систем
НАН України.
Кількість наукових публікацій в
українських виданнях – 36.
Кількість наукових публікацій в
зарубіжних виданнях – 12.
http://orcid.org/0000-0002-4700-6704,

Бекетов Олексій Геннадійович,
молодший науковий співробітник
Інституту програмних систем
НАН України.
Кількість наукових публікацій в
українських виданнях – 11.
Кількість наукових публікацій в
зарубіжних виданнях – 1.
http://orcid.org/0000-0003-4715-5053.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559.
E-mail: doroshenkoanatoliy2@gmail.com,
oayat@ukr.net,
beketov.oleksii@gmail.com