УДК 681.3

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

ФОРМАЛІЗОВАНЕ ПРОЕКТУВАННЯ ТА СИНТЕЗ ПАРАЛЕЛЬНИХ ПРОГРАМ ДЛЯ ВІДЕОГРАФІЧНИХ ПРИСКОРЮВАЧІВ

Проведено налаштування алгеброалгоритмічного інструментарію на формалізоване проектування та синтез програм, що використовують відеографічні прискорювачі. Продемонстрована ефективність використання багатопроцесорних графічних прискорювачів для обчислювальних задач на прикладі спеціально розробленого паралельного алгоритму чисельного інтегрування задачі N тіл.

Вступ

Сучасна обчислювальна техніка досягла того рівня розвитку, коли подальше зростання продуктивності за рахунок зменшення розміру кристалів ускладнюється природними перешкодами. Тому останнім часом розвитку набувають технології, що використовують масивно-паралельні блоки обчислювачів; ПК на основі багатоядерних процесорів вже давно набули побутового поширення.

В 2006 році розробник графічних прискорювачів NVIDIA презентував GPGPU технологію CUDA (Compute Unified Device Architecture), що дозволяє проводити обчислення використовуючи графічні прискорювачі [1–3]. Маючи виробничу продуктивність порядку сотень гігафлопс, графічні прискорювачі надають змогу проводити об'ємні обчислення навіть на звичайному ПК.

Незважаючи на наявність спеціалізованих засобів CUDA, розробка програм для графічних прискорювачів (Graphics Processing Unit, GPU) залишається трудомісткою задачею, яка вимагає від розроблювача знання низькорівневих деталей апаратної і програмної платформи. Тому актуальною є задача автоматизації процесу розробки. У даній роботі запропонований розвиток формальних методів проектування, що ґрунтується на концепції алгебри алгоритмів [4] з використанням техніки переписувальних правил [5] для автоматизованої розробки ефективних програм для графічних прискорювачів. Дана робота продовжує дослідження, розпочаті в [4, 6, 7], з автоматизації процесу проектування і розробки ефективних паралельних програм. Особливість даної роботи – використання нової апаратної платформи для паралельних обчислень, графічних прискорювачів, що дозволяє досягти значного підвищення продуктивності в порівнянні з використанням багатоядерних процесорів загального призначення.

У даній статті виконано налаштування розробленого у попередніх роботах Інтегрованого інструментарію проектування та синтезу програм (ІПС) [4, 6] на автоматизацію конструювання паралельних програм, що використовують відеографічні прискорювачі. Ефективність використання ІПС та GPU продемонстрована на прикладі задачі *N* тіл. Згадана задача полягає в дослідженні поведінки системи матеріальних точок, в якій кожна точка неперервно взаємодіє з усіма іншими точками системи [8]. Найпростішим методом розрахунку в задачі *N* тіл є обчислення сил взаємодії попарно між усіма точками системи на дискретній часовій сітці. Для задачі *N* тіл кількість операцій, необхідних для здійснення одного кроку по часу, квадратично збільшується із зростанням кількості частинок. Велика кількість операцій з даними, які необхідно здійснити при розрахунках внутрішніх взаємодій системи, призводить до необхідності розробки алгоритмів прискорення обчислення взаємодій. Оскільки основна задача може бути повністю розщеплена на паралельно виконувані підзадачі, виникає можливість застосування паралельних алгоритмів обчислень [9, 10]. Якщо до розрахунків залучи-

ти кластер з кількістю обчислювальних вузлів *N*, і кожен вузол буде проводити розрахунки для окремої точки, обчислення будуть виконуватись одночасно, що дозволить суттєво заощадити розрахунковий час.

1. Система ІПС та її налаштування на розробку CUDA програм

Інструментарій ІПС ґрунтується на системах алгоритмічних алгебр (САА) В.М. Глушкова і методі діалогового конструювання синтаксично правильних програм [1]. Його особливість полягає в спільному використанні трьох форм подання алгоритму при його конструюванні: аналітичної (формула в алгебрі алгоритмів), природно-лінгвістичної (текстової) і графової (граф-схеми). Компонентами розробленого інтегрованого інструментарію [1] є діалоговий конструктор синтаксично правильних програм (ДСП-конструктор), редактор граф-схем, генератор САА-схем і база даних. У ДСП-конструкторі здійснюється проектування алгоритмів у природно-лінгвістичній (САА-схема) і алгебраїчній (регулярна схема) формах. При цьому конструювання алгоритму виконується порівнево зверху вниз за допомогою суперпозиції операторних і предикатних операцій САА і подане у вигляді дерева. База даних ІПС призначена для зберігання, введення і редагування операцій САА в алгебраїчній та текстовій формах, а також містить шаблони реалізацій операцій цільовими мовами програмування (C, C++, Java). За схемами алгоритмів та описами САА конструкцій (із бази даних) ДСП-конструктор виконує синтез програм.

З метою орієнтації ДСП-конструктора на проектування і генерацію програм для відеографічних прискорювачів, в БД ІПС були включені додаткові конструкції. Перш ніж розглянути згадані конструкції, наведемо спочатку стислий огляд моделі GPU в структурі CUDA.

CUDA (Compute Unified Device Architecture) [2, 3] — програмно-апаратна архітектура, розроблена компанією NVIDIA, що дозволяє проводити обчислення, використовуючи графічні процесори NVIDIA, що підтримують технологію GPGPU (General Purpose computing on Graphics Processing Units). CUDA SDK дозволяє програмістам реалізовувати на спеціальному спрощеному діалекті мови програмування C алгоритми, виконувані на графічних процесорах NVIDIA, та включати спеціальні функції в текст програми на C. CUDA дає розробнику можливість самостійно організовувати доступ до набору інструкцій графічного прискорювача та керувати його пам'яттю, організовувати складні паралельні обчислення.

В моделі CUDA графічний прискорювач розглядається як спеціальний пристрій, що є масивно-паралельним співпроцесором центрального пристрою, має власну пам’ять та здатен одночасно виконувати велику кількість підпрограм – потоків. При виконанні програма на CUDA використовує як центральний пристрій, так і графічний. Типова схема виконання програми наступна:

– виділення області пам’яті на GPU та копіювання даних з CPU у виділену область пам’яті GPU;

– запуск ядра – паралельної частини програми, що виконується на GPU. Запуск виконує та керує ним CPU;

– копіювання отриманих результатів з пам’яті GPU до CPU та очищення виділеної пам’яті.

Основний процес CUDA працює на головному пристрої. Код для CPU робить наступне: ініціалізує GPU, розподіляє пам'ять на відеокарті і системі, копіює вихідні дані в пам'ять відеокарти, здійснює запуск ядер, копіює отримані результати з відеопам'яті, звільняє пам'ять і завершує роботу.

Апаратно графічні прискорювачі NVIDIA, що підтримують технологію CUDA, складаються з деякої кількості CUDA-ядер, кожне з яких здатне одночасно виконувати певну кількість потоків. Усі потоки підпорядковуються наступній ієрархії. Верхній рівень ієрархії – сітка – підпорядковує усі потоки, що виконують ядро. Сітка являє собою одно- або двовимірний масив блоків. Кожен блок – це одновимірний або двовимірний масив потоків, причому всі блоки, що утворюють сітку,

мають однакові розмірність та розмір. Звертання до окремих потоків відбувається за допомогою індексів: кожен блок у сітці має адресу (індекс блоку у сітці), аналогічно кожен потік у блоці має свій власний індекс всередині блоку; таким чином, кожний потік має унікальний ідентифікатор. Потоки можуть взаємодіяти між собою лише всередині одного блоку; під взаємодією розуміється використання окремої для кожного блоку так званої спільної пам'яті, а також синхронізація потоків, що може бути здійснена між потоками окремого блоку, проте не може бути здійснена на всьому GPU. Програма GPU (ядро) виконується над сіткою блоків потоків.

Таким чином, розділяючи основну задачу на сукупність підзадач, що можуть виконуватись незалежно одна від одної, і розв'язуючи ці підзадачі, використовуючи одночасно виконувані потоки, досягається паралелізм виконання алгоритму.

З метою налаштування системи ІПС на конструювання та генерацію програм для GPU в сигнатуру САА, а також в БД ІПС, було додано такі операції:

- операції взаємодії з відеопам'яттю, а саме: виділення і очищення пам'яті GPU; копіювання даних із пам'яті CPU в пам'ять GPU і у зворотному напрямку. Наприклад, текст оператору виділення пам'яті GPU має вигляд:

```
"Виділити пам'ять для змін-
ної (var) розміром (size) в
відеопам'яті"
```

- операція запуску ядра:

```
ЗапускЯдра (blocksPerGrid,
            threadsPerBlock)
(
   "оператор"
)
```

де `blocksPerGrid` – кількість блоків у сітці; `threadsPerBlock` – кількість потоків у кожному блоці; `"оператор"` – частина алгоритму, що буде виконуватися паралельно потоками CUDA;

- синхронізатор – операція, що здійснює очікування завершення обчислень усіма потоками CUDA:

```
ЧЕКАТИ 'Обробка у всіх
потоках закінчена'
```

Застосування цих операцій у даній роботі продемонстроване на прикладі проектування паралельного алгоритму для розв'язання задачі *N* тіл. У розділі 2 розглянуті загальна постановка задачі та використовуваний метод чисельного інтегрування. У розділі 3 наведено опис та САА-схеми послідовної та паралельної реалізацій алгоритму.

2. Гравітаційна задача *N* тіл та чисельне інтегрування

2.1. Постановка задачі. Задача багатьох тіл, що взаємодіють між собою під дією гравітаційних сил, є класичною задачею ньютонівської механіки, і є випадком задачі *N* тіл [8]. Задача розв'язна аналітично у випадку двох тіл, у випадку трьох тіл розв'язки побудовано для часткових випадків початкових даних та у вигляді збіжних рядів для загального випадку, а також доведено, що загальний розв'язок неможливо виразити через алгебраїчні чи трансцендентні функції швидкостей та координат.

Розглянемо систему *N* матеріальних точок з відомими масами m_i, що попарно взаємодіють між собою згідно закону тяжіння Ньютона. Нехай положення та швидкості тіл в початковий момент часу $t = 0$ відомі і становлять $r_i\big|_{t=0} = r_0$ та $v_i\big|_{t=0} = v_0$ відповідно. Необхідно наближено знайти положення та швидкості в наступні моменти часу.

Еволюція системи *N* матеріальних точок описується системою рівнянь

$$\frac{dr_i}{dt} = v_i,$$

$$\frac{dv_i}{dt} = \sum_{j \neq i} G m_j \frac{r_j - r_i}{\left|r_j - r_i\right|^3},$$

де $i = \overline{1, N}$, G – гравітаційна стала. Таким чином, задача зводиться до інтегрування системи з $2N$ диференціальних рівнянь першого порядку.

2.2. Чисельне інтегрування. Проінтегруємо подану задачу чисельно з дискретизацією за часовою змінною. Застосуємо метод типу предиктор-коректор з використанням інтерполяційних многочленів Ерміта [11]. Оберемо досить малий крок часу Δt. Інтегрування відбувається за наступною схемою.

1. Для кожної точки, виходячи з відомих положень точок системи та їх швидкостей в момент часу t, обчислюються сила, що діє на неї з боку інших точок, нормована на масу даної частинки, та її похідна:

$$a_{0,i} = -\sum_{j\neq i} Gm_j \frac{r_{ij}}{(r_{ij}^2+\varepsilon^2)^{3/2}}, \qquad (1)$$

$$\dot{a}_{0,i} = -\sum_{j\neq i} Gm_j \left(\frac{v_{ij}}{(r_{ij}^2+\varepsilon^2)^{3/2}} + \frac{3(v_{ij}\cdot r_{ij})r_{ij}}{(r_{ij}^2+\varepsilon^2)^{5/2}} \right), \qquad (2)$$

де

$$r_{ij} = r_j - r_i,\ v_{ij} = v_i - v_j,$$

а ε – пом'якшуючий параметр, необхідність введення якого зумовлена дискретністю часової сітки.

2. З урахуванням отриманих значень $a_{0,i}$ та $\dot{a}_{0,i}$ обчислюються положення точок системи в момент часу $t+\Delta t$:

$$r_{p,i} = \frac{\Delta t^3}{6}\dot{a}_{0,i} + \frac{\Delta t^2}{2}a_{0,i} + \Delta t v_{0,i} + r_i, \qquad (3)$$

$$v_{p,i} = \frac{\Delta t^2}{2}\dot{a}_{0,i} + \Delta t a_{0,i} + v_i. \qquad (4)$$

3. З урахуванням нових положень точок системи обчислюються сила $a_{1,i}$ та її похідна $\dot{a}_{1,i}$, що діє на точку в момент часу $t+\Delta t$, використовуючи формули, аналогічні формулам другого пункту:

$$a_{1,i} = -\sum_{j\neq i} Gm_j \frac{r_{ij}}{(r_{ij}^2+\varepsilon^2)^{3/2}}, \qquad (5)$$

$$\dot{a}_{1,i} = -\sum_{j\neq i} Gm_j \left(\frac{v_{ij}}{(r_{ij}^2+\varepsilon^2)^{3/2}} + \frac{3(v_{ij}\cdot r_{ij})r_{ij}}{(r_{ij}^2+\varepsilon^2)^{5/2}} \right). \qquad (6)$$

4. Використовуючи інтерполяційний поліном Ерміта, апроксимуються друга та третя похідні сили, що діють на i-ту точку системи в початковий момент часу, виходячи із значень $a_{0,i}$, $\dot{a}_{0,i}$, $a_{1,i}$ та $\dot{a}_{1,i}$:

$$\ddot{a}_{0,i} = \frac{-6(a_{0,i}-a_{1,i}) - \Delta t(4\dot{a}_{0,i} + 2\dot{a}_{1,i})}{\Delta t^2}, \qquad (7)$$

$$\dddot{a}_{0,i} = \frac{12(a_{0,i}-a_{1,i}) + 6\Delta t(\dot{a}_{0,i} + \dot{a}_{1,i})}{\Delta t^3}. \qquad (8)$$

5. На останньому кроці алгоритму вводяться поправки для отриманих положень і швидкостей точок з урахуванням отриманих значень $\ddot{a}_{0,i}$ та $\dddot{a}_{0,i}$:

$$r_i = r_{p,i} + \frac{\Delta t^4}{24}\ddot{a}_{0,i} + \frac{\Delta t^5}{120}\dddot{a}_{0,i}, \qquad (9)$$

$$v_i = v_{p,i} + \frac{\Delta t^3}{6}\ddot{a}_{0,i} + \frac{\Delta t^4}{24}\dddot{a}_{0,i}, \qquad (10)$$

і таким чином, отримуються остаточні значення положень і швидкостей в момент часу $t+\Delta t$.

Складність наведеного алгоритму становить $O(N^2)$. Точність даних, отриманих в результаті застосування поданого алгоритму, можна оцінити виходячи із оцінки похибки імпульсу системи.

3. Послідовна та паралельна реалізації алгоритму

3.1. CPU реалізація. При проведенні обчислень з використанням центрального процесора використаний алгоритм (рис. 1), що в точності повторює алгоритм, описаний у розділі 2.

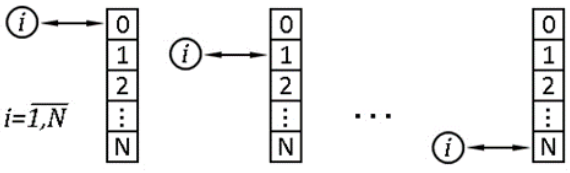

Рис. 1. Схема роботи послідовного алгоритму

Оскільки відеокарти персональних комп'ютерів оптимізовані під виконання 32-бітних обчислень, в послідовних обчисленнях на центральному процесорі також використовувався 32-бітний тип даних.

Узагальнена САА-схема послідовного алгоритму має вигляд:

```
СХЕМА "NBODY_SERIAL" ====

"Задати початкові значення мас,
координат та швидкостей частнок";
ДЛЯ (k ВІД 0 ДО STEPS - 1) ЦИКЛ
  ДЛЯ (i ВІД 0 ДО N - 1) ЦИКЛ
    "Обчислення прискорення час-
     тинок і його похідної (i)"
  КІНЕЦЬ ЦИКЛУ;
  ДЛЯ (i ВІД 0 ДО N - 1) ЦИКЛ
    "Виконати попередні обчислення
     координат та швидкостей (i)"
  КІНЕЦЬ ЦИКЛУ
  ДЛЯ (i ВІД 0 ДО N - 1) ЦИКЛ
    "Виконати обчислення поправок
     для координат та швидкостей (i)"
  КІНЕЦЬ ЦИКЛУ
  ДЛЯ (i ВІД 0 ДО N - 1) ЦИКЛ
    "Внести поправки для отриманих
     координат та швидкостей (i)"
  КІНЕЦЬ ЦИКЛУ
КІНЕЦЬ ЦИКЛУ

КІНЕЦЬ СХЕМИ "NBODY_SERIAL"
```

3.2. GPU реалізація. Основна ідея паралелізації алгоритму полягає у тому, що за проведення операцій з кожною окремою частинкою відповідає окремий потік GPU.

Існує ряд обмежень, яких слід дотримуватись при складанні алгоритму задля коректної роботи програми:

1) максимальна та мінімальна кількості потоків у блоці складають 512 та 64 одиниць відповідно, причому найбільша ефективність роботи ядер CUDA досягається, якщо використовувати 256 потоків на блок. Таким чином, кількість блоків *B*, що використовуються, обирається рівною

$$B = \begin{cases} \dfrac{N}{256}, & N \vdots 256, \\ \left[\dfrac{N}{256}\right] + 1, & N \not\vdots 256; \end{cases}$$

2) обмеження максимальної кількості операцій, що потік виконує за один запуск ядра. Виходячи з цього обмеження, при досить великих кількостях частинок, у потоках не можна використовувати цикли за повною кількістю частинок. В таких випадках необхідно розбити внутрішні цикли на підцикли та виконувати циклічний запуск ядра.

Керуючись наявними типами пам'яті з метою оптимального її використання і, таким чином, досягнення найбільш ефективної роботи GPU, розроблено алгоритм, що працює за наступною схемою.

1. Нехай частинки занумеровані та кожній із них співставлено відповідний потік. Кожен потік отримує початкові параметри своєї власної частинки, яку він буде обробляти.

2. За порядковим номером обирається поточна частинка, починаючи з першої. Координати, швидкості та маса частинки заносяться у спільну пам'ять кожного блоку потоків. Таким чином, потоки кожного блоку отримують змогу швидкого доступу до координат поточної частинки. Кожен потік здійснює обчислення внеску взаємодії між своєю та поточною частинками в суми (1) та (2). Крок повторюється, доки не будуть підраховані всі попарні взаємодії між частинками (рис. 2).

Рис. 2. Схема роботи паралельного алгоритму

3. Кожен потік проводить розрахунки попередніх значень координат та швидкостей згідно формул (3), (4).

4. Аналогічно другому пункту цієї схеми проводяться розрахунки за формулами (5) та (6).

5. Кожен потік вводить поправки до отриманих раніше попередніх значень положень та швидкостей згідно формул

(7), (8), (9), (10), і, таким чином, отримує остаточні результати.

Пункти 1–5 повторюються для кожного часового кроку, доки не будуть отримані значення в кінцевий момент часу.

Для переходу від послідовної САА-схеми алгоритму до паралельної виконано такі основні трансформації:

1) додано оператори взаємодії із відеопам'яттю;

2) у вкладених циклах за змінною i для першого та третього циклів виконано заміну *N* на N/512. В циклах обчислення проводяться не для всіх частинок одразу, а для наборів по 512, оскільки проведення обчислень для всіх частинок одразу або для наборів більшого розміру призводить до нестабільної роботи GPU;

3) вилучено другий та четвертий вкладені цикли за змінною i, і залишено лише оператори тіла даних циклів;

4) додано виклики операції ЗапускЯдра(blocksPerGrid, threads PerBlock), де кількість потоків у межах кожного блоку threadsPerBlock = = 256; кількість блоків blocksPerGrid = = N / threadsPerBlock;

5) вставлено виклики синхронізаторів.

Далі наведена отримана в результаті перетворень паралельна САА-схема.

```
СХЕМА "NBODY_PARALLEL" ====
"Задати початкові значення мас,
координат та швидкостей частинок";
"Виділити область пам'яті на GPU
та скопіювати дані з CPU в GPU";
ДЛЯ (k ВІД 0 ДО STEPS - 1) ЦИКЛ
  ДЛЯ (i ВІД 0 ДО N / 512 - 1) ЦИКЛ
    ЗапускЯдра (blocksPerGrid,
                threadsPerBlock)
    (
       "Обчислення прискорення
       частинок і його похідної (i)"
    );
    ЧЕКАТИ 'Обробка у всіх
            потоках закінчена'
  КІНЕЦЬ ЦИКЛУ;
  ЗапускЯдра (blocksPerGrid,
              threadsPerBlock)
  (
     "Виконати попередні обчис-
      лення координат та швидкос-
      тей"
  );
  ЧЕКАТИ 'Обробка у всіх потоках
         закінчена'
  ЗАТЕМ
  ДЛЯ (i ВІД 0 ДО N / 512 - 1) ЦИКЛ
    ЗапускЯдра (blocksPerGrid,
                threadsPerBlock)
    (
      "Виконати обчислення
      поправок для координат та
      швидкостей(i)"
    );
    ЧЕКАТИ 'Обробка у всіх
           потоках закінчена'
  КІНЕЦЬ ЦИКЛУ;
  ЗапускЯдра (blocksPerGrid,
              threadsPerBlock)
  (
     "Внести поправки для
      отриманих координат та
      швидкостей"
  );
  ЧЕКАТИ 'Обробка у всіх потоках
         закінчена'
КІНЕЦЬ ЦИКЛУ;
"Скопіювати отримані результати з
пам'яті GPU до CPU та очистити
виділену пам'ять"
КІНЕЦЬ СХЕМИ "NBODY_PARALLEL"
```

Для автоматизованої трансформації алгоритмів спільно з ІПС використовується система переписування термів TermWare [5]. Продемонструємо застосування правил TermWare для виконання згаданих вище перетворень (2) та (4). Наприклад, терм для першого вкладеного циклу із послідовного алгоритму має вигляд:

```
ForLoop(i, 0, Minus(N, 1),
   force0(i))
```

Для виконання вказаних двох трансформацій необхідно до даного терму застосувати правило

```
ForLoop($x1, $x2, Minus(N,1),
   $x3)  ->
```

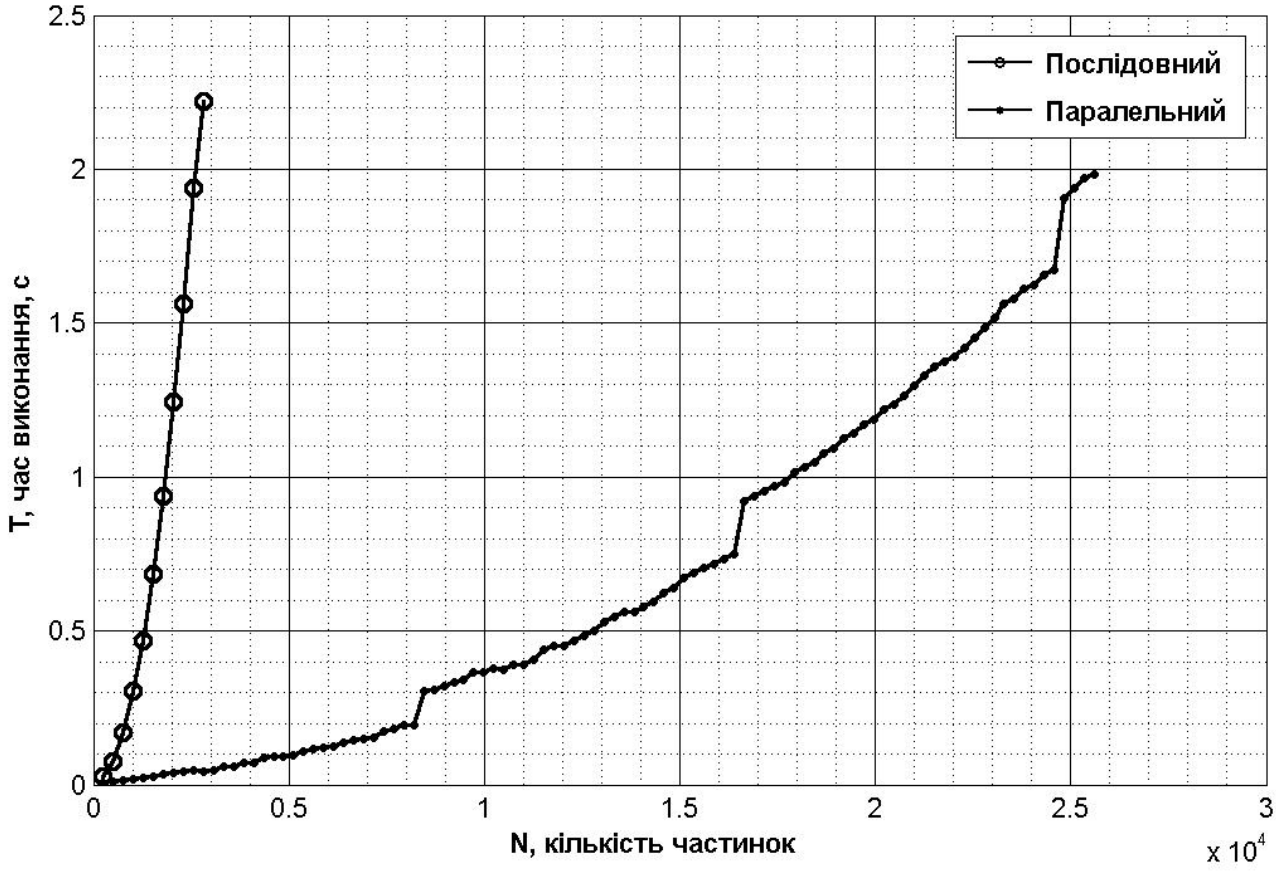

Рис. 3. Залежність часу виконання послідовного та паралельного алгоритмів від кількості досліджуваних частинок

```
ForLoop($x1, $x2,
        Minus(Div(N, 512), 1),
    CALL_GPU(blocksPerGrid,
             threadsPerBlock,
        $x3)),
```

де `$x1`, `$x2`, `$x3` – змінні, що використовуються у процесі трансформації.

В результаті отримаємо терм

```
ForLoop(i, 0,
        Minus(Div(N, 512), 1),
    CALL_GPU(blocksPerGrid,
             threadsPerBlock,
      force0(i)))
```

За наведеною САА-схемою в системі ІПС було виконано генерацію паралельної програми мовою C++ із використанням функцій CUDA. Результати виконання цієї програми на GPU наведено у наступному розділі.

4. Результати експерименту

Обчислення в експериментальній задачі для підрахунку часу проводились з наступними початковими умовами. В початковий момент часу частинки розміщені у вузлах тривимірної паралелепіпедоподібної області розміру $16\times16\times(N/256)$. Кількість частинок обиралась кратною 256. Стартові швидкості частинок задані довільним чином і нормовані на одиницю. Пом'якшуючий параметр ε обрано рівним 0.01, гравітаційну сталу одиничною, часовий крок $\Delta t = 0.25$.

Випробування проводились із використанням процесора i5-3570 (у 32-бітному режимі) та графічного прискорювача GeForce GTX 650 Ti, що має наступні характеристики:

- 768 stream-процесори (CUDA-ядра), базова частота 928MHz;

- обсяг глобальної пам'яті 1024Mb;
- базова частота пам'яті 5400MHz.

Час виконання вимірювався для кількості частинок від 256 до 16384 з кроком 256 частинок та при окремих значеннях N (32768, 65536) на один крок за часом. В таблиці представлена вибірка отриманих значень від 256 частинок з подвоєнням кількості. Тут N – кількість задіяних частинок, T_{CPU} (с) – час виконання послідовної програми, T_{GPU} (с) – час виконання паралельної програми, T_{CPU} / T_{GPU} – коефіцієнт прискорення.

Таблиця. Час виконання послідовної та паралельної програм

N	T_{CPU}, с	T_{GPU}, с	T_{CPU} / T_{GPU}
256	0.02797	0.00525	5.32761
512	0.0766	0.00844	9.07582
1024	0.3031	0.01907	15.8941
2048	1.2422	0.03843	32.3237
4096	5.375	0.0703	76.4580
8192	32.094	0.1953	164.331
16384	161.453	0.75	215.270
32768	718.875	1.5516	463.312
65536	3045.22	6.0585	502.636

Низький рівень прискорення при невеликих значеннях N пояснюється неповною завантаженістю відеокарти, через що не досягається максимальна ефективність. Крім того, суттєвий вплив має те, що час, витрачений на проведення кроку алгоритму, складається, окрім часу, витраченого на проведення обчислень, з часу, витраченого на передачу даних з CPU на GPU та в зворотному напрямку.

На графіку, поданому на рис. 3, зображено залежність часу виконання кроку алгоритму від кількості досліджуваних матеріальних частинок для CPU та GPU програм.

Висновки

Проведено налаштування розробленого у попередніх роботах алгеброалгоритмічного інструментарію на формалізоване проектування та синтез програм, що використовують відеографічні прискорювачі. Продемонстрована ефективність використання ІПС та GPU для обчислювальних задач на прикладі спеціально розробленого паралельного алгоритму чисельного інтегрування задачі N тіл. Для проектування алгоритму використана мова САА-схем, перевагою якої є простота в навчанні і використанні, а також незалежність від мови програмування і можливість перекладу в довільну цільову мову. Перевагою застосування системи ІПС є також використання методу конструювання синтаксично правильних програм, який виключає можливість появи синтаксичних помилок у процесі проектування алгоритмів. Проведено експеримент з виконання згенерованої за допомогою ІПС паралельної програми на відеографічному прискорювачі.

1. *Harris M.* GPGPU: General-Purpose Computation on GPUs. – In SIG-GRAPH 2005 GPGPU COURSE, 2005.
2. NVIDIA. CUDA Programming Model Overview, 2008. http://www.sdsc.edu/us/training/assets/docs/NVIDIA-02-BasicsOfCUDA.pdf
3. *Боресков А.В., Харламов А.А.* Основы работы с технологией CUDA. – ДМК-Пресс, 2010. – 232 с.
4. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 631 с.
5. *Дорошенко А.Е., Шевченко Р.С.* Система символьных вычислений для программирования динамических приложений // Проблеми програмування. – 2005. – № 4. – С. 718–727.
6. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Формализованное проектирование эффективных многопоточных программ // Про-

блеми програмування. – 2007. – № 1. – С. 17–30.
7. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Об оценке сложности и координации вычислений в многопоточных программах // Проблеми програмування. – 2007. – № 2. – С. 41–55.
8. *Aarseth S.J.* Gravitational N-body simulations. – Cambridge University Press, 2003. – P. 1–17.
9. *Nyland L., Harris M., Prins J.* Fast N-body simulation with CUDA. – In GPU Gems 3, Addison-Wesley Professional, 2007. – P. 677–695.
10. *Zwart S., Belleman R., Geldof P.* High Performance Direct Gravitational N-body Simulations on Graphics Processing Units // New Astronomy. — 2007. – Vol. 12, Issue 8. – P. 641–650.
11. *Makino J., Aarseth S.J.* On a Hermite integrator with Ahmad-Cohen // Publications of the Astronomical Society of Japan, 44. – 1992. – P. 141–151.

Одержано 19.11.2012

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень,

Бекетов Олексій Геннадійович,
аспірант,

Жереб Костянтин Анатолійович,
кандидат фізико-математичних наук,
науковий співробітник,

Яценко Олена Анатоліївна,
кандидат фізико-математичних наук,
старший науковий співробітник.

Місце роботи авторів:

Інститут програмних систем
НАН України.
Тел.: (044) 526 3559,
E-mail: dor@isofts.kiev.ua
beketov.oleksii@gmail.com
zhereb@gmail.com
oayat@ukr.net