Про деякі способи розпаралелення алгоритмів розв'язування систем лінійних алгебраїчних рівнянь на гібридних комп'ютерах з графічними прискорювачами

Хіміч О.М., Баранов А.Ю., Чистякова Т.В.

Інститут кібернетики ім. В.М. Глушкова НАН України, Пр. Глушкова 40, м. Київ, Україна

Анотація. *Комп'ютери гібридної архітектури, які поєднують обчислення на багатоядерних комп'ютерах з прискоренням обчислень на графічних процесорах, дають можливість значно прискорити процес розв'язування задач лінійної алгебри. Проте виникають проблеми ефективної реалізації паралельних алгоритмів для гібридного комп'ютера, необхідність у плануванні обчислень на обчислювальних ресурсах CPU і GPU з метою якнайшвидшого отримання розв'язку задачі, оптимізації комунікаційних витрат між ядрами CPU і процесорами GPU. В роботі розглядаються деякі підходи створення алгоритмів та програм розв'язування систем лінійних алгебраїчних рівнянь для комп'ютерів гібридної архітектури при використанні сучасного високопродуктивного програмного забезпечення, яке дає можливість спростити процес програмування і при цьому досягти високу швидкодію. Подаються результати комп'ютерних досліджень з проблем, що розглядаються.*

Ключові слова

Комп'ютери гібридної архітектури, StarPU, бібліотека програм MKL, технологія програмування CUDA.

1 Вступ

В даний час спостерігається певна тенденція в розвитку обчислювальних систем. З одного боку продовжується зростання продуктивності комп'ютерів за рахунок збільшення кількості процесорних ядер комп'ютера, а з іншого боку, стають все більш популярними гібридні системи, гетерогенні системи, висока продуктивність яких обумовлена застосуванням обчислювальних ресурсів принципово нової архітектури.

Перший напрямок розвитку передбачає паралелізм алгоритмів розв'язування задач на однотипних процесорних ядрах, який на програмному рівні реалізується за допомогою спеціальних систем паралельного програмування таких, наприклад, як MPI.

Другий напрямок вимагає від алгоритму більш складної багаторівневої паралельної моделі, яка враховує різні архітектури обчислювальних ресурсів, що використовуються.

Розробка ефективних алгоритмів та програм для розв'язування задач на гібридних комп'ютерах вимагає як знань різних технологій програмування, наприклад, MPI, CUDA, так і особливостей гібридних архітектур.

В даній роботі розглядаються деякі підходи при розробці алгоритмів та програм для розв'язування систем лінійних алгебраїчних рівнянь (СЛАР) на гібридних комп'ютерах з графічними прискорювачами, які дають можливість дещо спростити процес програмування і при цьому досягти високу швидкодію комп'ютерних алгоритмів та програм.

Незважаючи на велику увагу до створення програмного забезпечення з цієї проблематики на різні типи комп'ютерних архітектур, багато проблем ефективного його використання залишаються.

Перш за все, при використанні сучасного програмного забезпечення для розв'язування СЛАР виникають проблеми розв'язування задач з наближено заданими вихідними даними, що потребує самостійного дослідження відповідності визначених властивостей комп'ютерної моделі задачі обраному алгоритму розв'язування, а також аналізу достовірності одержуваних результатів: оцінки спадкової похибки в наслідок похибки у вхідних даних та оцінки обчислювальної похибки [1].

При розробці ефективних паралельних алгоритмів для гібридних систем або при використанні готового програмного забезпечення для розв'язування СЛАР необхідно також самостійно виконати такі етапи робіт:

- декомпозицію задачі, тобто можливість розподілу задачі на підзадачі, які можуть бути реалізовані в значній мірі незалежно одна від одної;
- виділення для створеного набору підзадач інформаційні взаємозвязки, які повинні здійснюватися в ході розв'язування задачі;
- визначення який обчислювальний пристрій центральний процесор (CPU) або графічний прискорювач (GPU) є найбільш ефективним для реалізації підзадачі.

Дослідження показали, що найбільш ефективними виявляються блочні алгоритми розв'язування задач, які забезпечують збалансовану обробку матриць на комп'ютерах гібридної архітектури, ефективне використання кеш-пам'яті. Крім того, реалізацію цих алгоритмів можна звести до обмеженої кількості базових операцій лінійної алгебри, які можна оптимізувати під задану архітектуру, а також використати вже відомі бібліотеки стандартних програм, наприклад BLAS [2], CUBLAS [3], LAPACK [4] та ін.

Дослідження деяких стилів програмування проводилось на багатоядерній інтелектуальній робочій станції гібридної архітектури (з графічними прискорювачами) серії Інпарком, спільної розробки Інституту кібернетики ім. Глушкова НАН України та Державного науково-дослідного підприємства «Електронмаш». Розглядалися алгоритми та програми розв'язування СЛАР з щільними невиродженими матрицями з використанням одного CPU та одного GPU. Програма, що реалізує алгоритм розв'язування задачі, складається з програмних кодів для CPU, написаних на Сі або C++, та кодів для GPU, написаних з використанням технології CUDA.

2 Використання системи планування обчислень StarPU при розробці програм для розв'язування СЛАР на комп'ютерах гібридної архітектури

Система планування обчислень StarPU [5] дає можливість планувати виконання задачі на ядрах CPU та процесорів GPU в різних співвідношеннях, найбільш ефективних для конкретної задачі та алгоритму, що її реалізує. Крім того, для реалізації алгоритмів розв'язування СЛАР можна використати високопродуктивні бібліотеки програм з лінійної алгебри BLAS, LAPACK, CUBLAS.

Паралелізм задачі за допомогою системи STARPU організується, використовуючи послідовні коди програм, на основі планування обчислень частин алгоритму (підзадач), які повинні реалізуватися на тому чи іншому обчислювальному пристрої гібриду: тільки на CPU, тільки на GPU, з використанням обох пристроїв. StarPU аналізує та динамічно планує обчислення з урахуванням затрат часу виконання кожної підзадачі на кожному доступному обчислювальному пристрої. При цьому автоматично виконується асинхронне копіювання необхідних даних як між CPU та GPU, так і між окремими процесорами GPU. Вихідні дані задачі один раз реєструються в середовищіе StarPU, а в подальшому система сама копіює необхідні дані між CPU та GPU, максимально перекриваючи час виконання комутаційних зв'язків виконанням математичних операцій.

Планування обчислень включає:

1) реалізацію розв'язування підзадач на STARPU_CPU | STARPU_CUDA|(Виконання задачі тільки на CPU, на ядрах CPU з використанням GPU або тільки на GPU);

2) опис задачі функціями codelets та task для кожної архітектури;

3) автоматичний (StarPU) або ручний розподіл задачі на підзадачі (starpu_data_partition, starpu_data_filter і т.д.) та встановлення між ними залежностей, пріоритетів та порядку виконання (автоматично або вручну);

4) автоматичну побудову необхідної топології комп'ютера з оптимальної кількості процесів, автоматичний розподіл даних та копіювання массивів даних між CPU та GPU (struct starpu_sched_policy_s, schedulete the codelet і т.д.);

5) автоматичне планування обчислень з урахуванням часу виконання задачі, вводу / виводу на різних технологічних пристроях з метою вибору найкращої продуктивності моделі (struct starpu_data_interface_ops_t, struct starpu_sched_policy_s, static struct starpu_perfmodel_t і т.д.;

6) теоретичну оцінку знизу часу виконання всієї задачі: starpu_bound_start, starpu_bound_stop та визначення оптимізованого мінімуму часу виконання всієї задачі.

Таким чином, StarPU за допомогою набору планувальників дає можливість досягнути максимально швидкої реалізації алгоритму в залежності від архітектури гібридної системи. Наприклад, для алгоритму LL^T-розвинення найбільш ефективним є Heft-планувальник, що базується на Heterogeneous Earliest Finish Time (HEFT) [5]. Для кожного доступного обчислювального пристрою цей планувальник підраховує величину Avail(Pi), яка визначає в який час кожний з обчислювальних пристроїв може бути доступним, тобто всі підзадачі завершаться. Причому, нова підзадача *T* буде виконуватися на тому обчислювальному пристрої, який мінімізує виконання задачі.

В залежності від очікуваного часу виконання $Est_{P_i}(T)$, цю умову можна визначити за такою формулою:

$$\min_{P_i}(Avail(P_i) + Est_{P_i}(T)).$$

Розробнику програми небхідно задати функції, які повертають значення очікуваного часу виконання задачі $Est_{P_i}(T)$. Для операцій, що розглядаються, оцінка часу виконання задачі на конкретному пристрої може бути легко отримана, оскільки на кожному кроці конкретного алгоритму час виконання операції залежить лише від порядку матриці. StarPU надає структуру starpu perfmodel t, до якої входять функції, що обчислюють час розв'язування підзадач. Таким чином, ми маємо можливість заздалегідь задавати модель для визначення часу виконання задачі. Також в StarPU можна задавати модель, яка базується на історії запуску задачі на попередніх кроках. Наприклад, у випадку реалізації алгоритму LL^T-розвинення, час виконання факторизації кожного діагонального блоку можна брати таким, яким він був на першому кроці.

Розглянемо, як можна реалізувати блочний алгоритм LL^T-розвинення на гібридній архітектурі з плануванням обчислень на CPU та GPU за допомогою системи StarPU.

За основу обчислювальної схеми беремо схему реалізації блочного алгоритму LL^T.-розвинення на основі послідовних кодів чотирьох операцій із бібліотек LAPACK та BLAS [2, 4]:

DPOTRF – розвинення ведучого діагонального блоку матриці;

DSYRK – модифікація діагональних (нижніх трикутних) блоків матриці;

DGEMM – обчислення блоків матриці, які знаходяться під діагоналлю справа від ведучого блоку;

Використовуючи відповідні програми з бібліотек BLAS (DTRSM, DSYRK, DGEMM) та LAPACK (DPOTRF) псевдокод можна представити у вигляді:

```
for (k = 0; k  < Nt; k++)
A[k][k] <– DPOTRF(A[k][k])
for (m = k+1; m  < Nt; m++)
A[m][k] <– DTRSM(A[k][k],A[m][k])
for (n = k+1; n  < Nt; n++)
A[n][n] <– DSYRK(A[n][k],A[n][n])
for (m = n+1; m  < Nt; m++)
A[m][n] <– DGEMM(A[m][k],A[n][k], A[m][n])
```

Як ми бачимо з псевдокоду, операція у другому рядку може бути виконана раніше, ніж був повністю виконаний попередній крок в циклі. Для початку її виконання достатньо, щоб на попередньому кроці було визначено A[k][k]. Аналогічні міркування можна привести і для інших операцій.

Таким чином, щоб приступити до виконання операцій на кроці k-циклу, не обов'зково, щоб крок k-1 був завершений. Достатньо, щоб були отримані необхідні для кожної конкретної операції блоки матриці. StarPU дає можливість заздалегідь оголосити всі операції, які будуть потрібні для виконання алгоритму, а також визначити між ними залежності, які дають можливість контролювати порядок виконання операцій. І потім всі операції будуть виконуватися в тому порядку і на тій кількості процесорів, які забезпечать найвищу ефективність реалізації алгоритму.

Приведемо псевдокод гібридної реалізації блочного алгоритму LU-факторизації :

```
for k = 1; … , p do
  DPGETRF(A_kk);
  cudaCopyDevice(A_kk)
  for i = k+1; … , p do
    cudaDTRSM(A_kk, A_ik);
  end for;
  for i = k+1; … , p do
    cudaDSYRK(A_ik, A_ii);
    cudaCopyHostAsync(A_kk);
    for j = k+1; … , p &!= i do
        cudaDGEMMAsync(A_ik, A_jk, A_ij);
    end for;
  end for;
end for;
```

В результаті, алгоритм складається з виконання наступних процедур:

- розподіл задачі на підзадачі;
- паралельні підзадачі (cudaDTRSM, cudaDSYRK, cudaDGEMM) "плануються" для ефективного виконання на GPU з урахуванням асинхронності та величини блоку;
- послідовна підзадача DGETRF – розвинення діагонального блоку виконується на CPU;

- малі задачі на CPU частково покриваються великими задачами на GPU (множення матриць).

Тут після знаходження діагонального блоку з індексами (k+1, k+1) всі інші функції запускаються асинхронно, що дає можливість перейти на наступний крок циклу, до завершення всіх операцій на GPU, та виконувати паралельно роботу на CPU.

З псевдокоду видно, що дані для виконання функції cudaDSYRK((A_{ik},A_{ii}) будуть отримані до повного завершення другого циклу.

Для ефективного виконання алгоритму потрібно, щоб кожна з функцій виконувалась в той момент, коли всі дані, котрі їй потрібні, були отримані. Отже, нам потрібно динамічно визивати функції в той момент, коли є можливість для їх виконання.

StarPU представляє схему реалізації блочного алгоритму LL^T-розвинення у вигляді направленого графу, вершинами якого будуть підпрограми (функції), що використовуються в псевдокоді, а ребра зв'язують ці вершини тоді і тільки тоді, коли для виконання вершини Б необхідні дані, які обчислюються у вершині А вже отримано. Таким чином, процес реалізації LL^T- розвинення має вигляд направленого ациклічного графу, в якому вершинами є підзадачі, які необхідно виконати, а направлені ребра – залежності між ними. Для кожної наступної підзадачі необхідні дані будуть готові тоді і тільки тоді, коли будуть виконані всі попередні підзадачі, від яких вона залежить. (див. рис. 1).

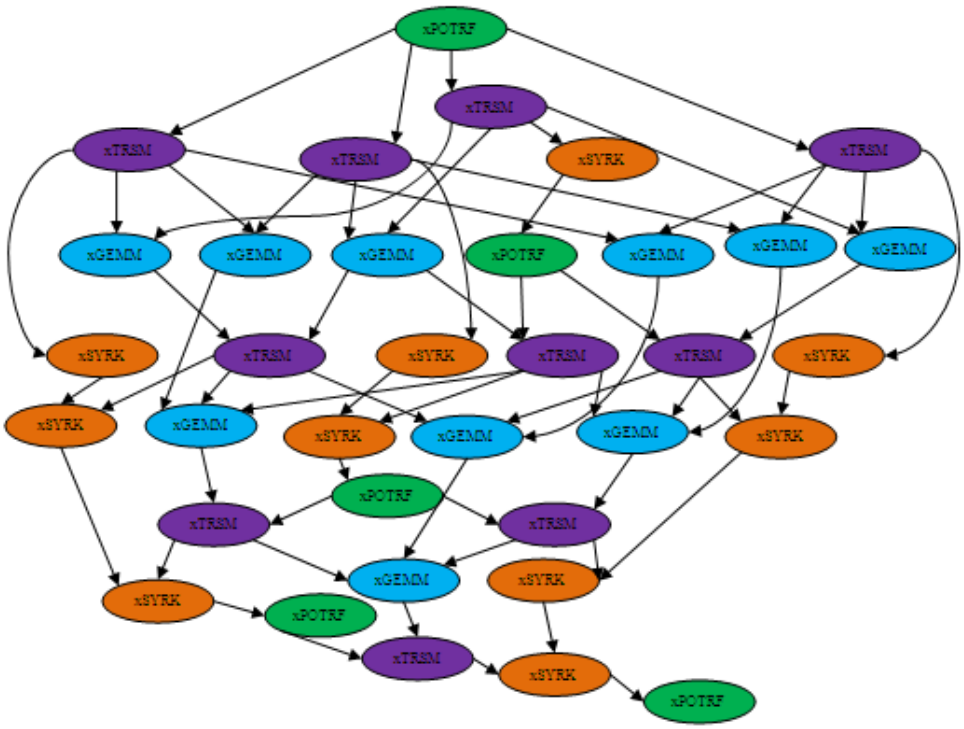

Рис. 1. Алгоритм LL^T-розвинення вигляді направленного ациклічного графу

Необхідно відмітити, що при реалізації блочних алгоритмів розв'язування СЛАР важливо вибрати величину блоку узгодженою з кеш-пам'яттю комп'ютера. Це значно прискорить виконання алгоритму.

Тому було проведено дослідження оптимальної величини блоку для алгоритмів, що розглядаються. Найкращі за часом результати ми одержали при величині блоку 256. Це було враховано при розв'язуванні систем на гібридній архітектурі комп'ютера з використанням системи StarPU.

Порівняльну характеристику часу (сек) розв'язування СЛАР з симетричними додатньо означеними матрицями блочним LL^T-алгоритмом тільки на CPU та при плануванні обчислень за допомогою StarPU на CPU з використанням графічного прискорювача представлено в табл.1.

Табл. 1. Порівняння часу реалізації LL^T-алгоритму на CPU+GPU (з використанням StarPU) та CPU

Порядок матриці	***Обчислювальний пристрій***	***Час розв'язування***
4096	CPU+GPU	4,5
	CPU	66,27
8192	CPU+GPU	5,8
	CPU	499,64

Аналогічно можна реалізувати розв'язування СЛАР з щільними невиродженими матрицями *LU*-алгоритмом та з виродженими або прямокутними матрицями, використовуючи *SVD*-розвинення матриць.

Наприклад, *LU*-алгоритм реалізується за такою схемою:

- розподіл задачі на підзадачі;
- паралельні підзадачі (cudaDSWAP, cudaDGER, cudaDLASWP "плануються" для ефективного виконання на GPU з урахуванням асинхронності та величини блоку;
- послідовна підзадача DGETRF – розвинення діагонального блоку виконується на CPU;
- малі задачі на CPU частково покриваються великими задачами на GPU (множення матриць).

Порівняння часу (в сек) розв'язування СЛАР з щільними невиродженими матрицями *LU*-алгоритмом тільки на CPU та при плануванні обчислень за допомогою StarPU на CPU з використанням графічного прискорювача представлено в табл. 2.

Табл. 2. Порівняння часу реалізації *LU*-алгоритму на CPU+GPU (з використанням StarPU) та на CPU

Порядок матриці	***Обчислювальний пристрій***	***Час розв'язування***
4096	CPU+GPU	6,15
	CPU	60,03
8192	CPU+GPU	22,00
	CPU	470,12

Як показали результати дослідження, які наведено в табл. 1-2, час виконання алгоритмів розв'язування СЛАР з щільними матрицями на CPU+GPU при плануванні обчислень за допомогою StarPU зменшується в десятки разів у порівнянні з часом виконання лише на CPU.

3 Використання бібліотеки програм MKL при розробці програм для розв'язування СЛАР на комп'ютерах гібридної архітектури

Бібліотека Intel MKL (Intel Maths Kernel Library) підтримує розпаралелювання і оптимізована під багатоядерні системи [6]. Використовуючи Intel MKL, розробник може підвищити продуктивність програм, коли доцільно використання багатопоточність.

Одна з особливостей Intel MKL – незалежність від компілятора. Це означає, що програмний код, написаний одного разу для однієї системи, вільно переноситься на інші системи.

Бібліотека Intel MKL має в своєму арсеналі більш гнучкі у порівнянні з іншими бібліотеками підходи для розв'язування тієї чи іншої задачі. Для розв'язування СЛАР в бібліотеці MKL є наявним набір функцій, який найбільше підходить в даному конкретному випадку: буде проведено пошук найбільш оптимального шляху для кожного типу матриці лінійної системи.

Набір програмного забезпечення в бібліотеці Intel MKL серії 10, яка функціонує на інтелектуальній робочій станції Інпаркм-G:

- BLAS – функції для однопоточного виконання матрично-векторних операцій;
- LAPACK – функції для однопоточних та багатопоточних програмних модулів розв'язування СЛАР та алгебраїчної проблеми власних значень блочними алгоритмами (використовується OpenMP);
- ScaLAPACK, BLACS, PBLAS – функції для розв'язування задач лінійної алгебри на комп'ютерах з розподіленою пам'яттю (використовується система MPI) на основі відповідних програм LAPACK;
- Sparse Solver – функції для розв'язування систем с симетричними та симетрично структурованими розрідженими матрицями ітераційними методами. За допомогою цих програм можна розв'язувати СЛАР з додатньо означеними та напівозначеними матрицями;
- Vector 0Math Library (VML) function – функції для обчислення основних математичних функцій;
- Vector Statistical Library (VSL) – функції для статистичних обчислень;
- Conventional FFTs and Cluster FFTs – функції перетворення Фур'є в тому числі і на кластерах;
- Partial Differential Equations support – деякі функції для використання при розв'язуванні диференціальних рівнянь.
- Trigonometric Transform routines, Fast Poisson, Laplace, and Helmholtz Solver (Poisson Library) routines – функції для швидкого обчислення синуса, косинуса та інших тригонометричних обчислень, інструменти для розв'язування рівнянь з частинними похідними (такими інструментами є, наприклад, обчислювачі тригонометричних функцій, процедури розв'язування рівнянь Пуасона).

Intel MKL версії 10 може використовуватися для розв'язування задач на комп'ютерах з єдиною та розподілею пам'яттю, з використаннм OpenMP (LAPACK) / MPI (ScaLAPACK) паралелізму. З однієї сторони

користувач може використовувати один MPI процес на вузлі з подальшим розпаралеленням на потоки за допомогою OpenMP, а з іншої сторони можна реалізувати лише MPI паралелізм. Для використання OpenMP без перекомпіляції, достатньо встановити змінну OMP_NUM_THREADS в додатнє значення.

Розглянемо, як можна реалізувати блочний алгоритм LL^T-розвинення на гібридній архітектурі з використанням бібліотек IntelMKL та CUBLAS.

Згідно схеми реалізації алгоритму, яку наведено в пункті 2, для факторизації діагональних підматриць будемо використовувати функцію DPOTRF з багатопоточного LAPACK бібліотеки Intel MKL (було розпаралелено на два потоки), а для модифікації інших блоків використаємо функції cublasDTRSM, cublasDSYRK, cublasDGEMM з бібліотеки CUBLAS (версія BLAS для графічного прискорювача). Час розв'язування задачі (сек.) для різних порядків матриць наведено в табл. 3.

Табл. 3. Час (сек.) реалізації LL^T -алгоритму на CPU+GPU з використанням Intel MKL та CUBLAS

Порядок матриці	***Час розв'язування***
1024	1,06
2048	2,34
4096	5,67
8192	29,23

Порівнюючи результати, які наведені в табл.3 та табл. 1 для відповідних порядків матриць, ми бачимо значне прискорення часу розв'язування задачі на архітектурі CPU+GPU при застосуванні Intel MKL та CUBLAS у порівнянні з часом виконання задачі на CPU. Проте слід зазначити, що таке прискорення одержано насамперед завдяки використанню графічного прискорення. А щоб одержати суттєвого прискорення часу виконання задачі з використанням бібліотеки Intel MKL, необхідно використати більше потоків. З іншої сторони, використання програм бібліотеки Intel MKL значно полегшує та прискорює час створення програм для гібридної архітектури.

4 Висновки

Використання графічних процесорів для розв'язування СЛАР з щільними матрицями показало суттєво зменшення часу розв'язування задач за рахунок використання графічних прискорювачів. Використання відомих високопродуктивних бібліотек програм CUBLAS, MKL StarPU не тільки сприяє спрощенню програмування для гібридних систем, але також значно зменшує час розв'язування задач. Таким чином, забезпечується висока продуктивність використання комп'ютерів гібридної архітектури.

Для ефективного виконання програм в розподіленому середовищі необхідно створювати такі алгоритми та програми, які добре масштабуються з урахуванням обмежень мережевих комунікацій: пропускні можливості мережі та затримки мають великий вплив на швидкодію алгоритмів та програм для гібридних архітектур. Автори зіткнулися з цією проблемою при розробці програм для гібридних архітектур, використовуючи технологію CUDA версії 3 та нижчих.

Добре розуміючи цю проблему, розробники технології CUDA з компанії NVIDIA приділяють велику увагу покращенню можливостей CUDA − кожні 3-5 місяців випускаються версії з новими покращеними можливостями.

На сьогоднішній день широко NVIDIA CUDA 4.0 [7], яка не тільки полегшує процес програмування, але також сприяє підвищенню швидкодії виконання завдань у порівнянні с попередніми версіями NVIDIA CUDA.

Графічний прискорювач є пристроєм, який реалізує потокову обчислювальну модель (stream computing model). Обробка елементів потоку даних здійснюється ядром (*kernel*). З появою CUDA GPU розглядається як спеціалізований пристрій (*device*), що є співпроцесором до CPU (*host*), має свою власну пам'ять, має можливість виконувати паралельно величезну кількість окремих ниток (*threads*), які об'єднуються в ієрархію − *grid/block/thread*.

Найпомітнішою зміною в CUDA 4.0 є перехід до єдиного адресного простору. Тепер пам'ять центрального процесора і всіх графічних прискорювачів можна адресувати одним єдиним вказівником. Для копіювання даних замість чотирьох параметрів копіювання (cudaMemcpyHostToHost, cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost) використовується один параметр (cudaMemcpyDefault).

Іншим корисним нововведенням стала технологія NVIDIA GPU Direct 2.0, яка забезпечує використання декількох GPU одним потоком з CPU − один потік з CPU хоста може мати доступ до всіх GPU в системі, а також забезпечується зв'язок між GPU в рамках одного сервера. У попередніх версіях CUDA при копіюванні з GPU-пам'яті завжди використовувався центральний процесор, що суттєво гальмувало весь процесс виконання завдання. Технологія GPU Direct дає можливість копіювати дані на GPU практично з будь-якого пристрою − жорсткий диск, інтерфейс InfiniBand або інший графічний прискорювач. CPU при цьому не використовується.

В жовтні 2012 року компанія Nvidia представила п'яту версію технології CUDA.

Серед нових можливостей CUDA 5.0 [8], можна відмітити такі:

- Динамічна паралельність, тобто обчислювальні потоки можуть динамічно породжувати нові. Бібліотека CUDA BLAS надає можливість розробникам програм застосовувати динамічний паралелізм для їх власних бібліотек, які можуть визиватися напряму з коду на GPU.
- Бібліотеки для графічних процесорів. Нова бібліотека CUDA BLAS дозволяє використовувати можливості динамічної паралельності з сторонніх бібліотек.
- GPUDirect — прямі зв'язки між графічними процесорами через шину PCI-E, прямий доступ до пам'яті між мережевими картами та GPU. Це дає змогу значно зменшити затримки виконання команд типу MPISendRecv між вузлами GPU гібридного комп'ютера та підвищити загальну швидкодію роботи програм.

Як показали деякі дослідження, вказані можливості останніх версій CUDA дають змогу значно зменшити час на мережеві комунікації при розв'язуванні задач на гібридних комп'ютерах. Крім того, вбудований редактор в CUDA 5.0 та конкретні приклади прискорюють генерування коду CUDA.

Література

[1] А.Н. Химич, И.Н. Молчанов, А.В. Попов, Т.В. Чистякова, М.Ф. Яковлев Параллельные алгоритмы решения задач вычислительной математики. – Киев: Наук. думка, 2008. – 248 с
[2] BLAS (Basic Linear Algebra Subprograms. http://www.netlib.org/blas/
[3] CUBLAS Linear Algebra. http://developer.download.nvidia.com/CUBLAS_Library.pdf
[4] LAPACK – Linear Algebra PACKage. http://www.netlib.org/lapack/
[5] C. Augonnet, S. Thibault, R. Namyst, P. Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. Concurrency and Computation: Practice and Experience, Euro-Par 2009 best papers issue. Accepted for publication, 2010.
[6] Intel Maths Kernel Library. http://software.intel.com/en-us/intel-mkl.
[7] CUDA TOOLKIT 4.0. // http://developer.nvidia.com/cuda-toolkit-4.0.
[8] CUDA 5. http://www.nvidia.ru/object/nvidia-cuda-5-parallel-computing-20121015-ru.html