Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Кафедра обчислювальної математики факультету кібернетики

**Паралельні обчислення у методі ‘сіток’ розв’язання еліптичних рівнянь на комп’ютері гібридної архітектури**

**Текстова частина до курсової роботи**

**за спеціальністю „Прикладна математика” 6.040301**

#### Керівник курсової роботи

доктор фіз.-мат. наук

Хіміч Олександр Михайлович

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2015 р.

Виконав студент

Оленченко Ілля Андрійович

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_ 2015 р.

Київ 2015

**TODO в текстову частину**

I) Текстова частина (ТЧ):

0) общее оформление http://web.znu.edu.ua/lab/fordep/oformlenie/diplom\_specialista.htm

1) прочитать как должно быть оформлена работа

2) індивідуальне завдання на КР(МР);

3) календарний план;

4) зміст;

5) анотація 1 ст. ;

6) вступ 1-3 ст.;

7) основна частина (її розділи): 30-50 ст.;

– аналіз існуючих методів (алгоритмів) вирішення поставленої задачі;

– обґрунтування вибору рішення;

– вибір принципу дії системи чи обґрунтування методик;

– розробка структурної і (або) функціональної схеми;

– розробка принципової схеми;

– експериментальні дослідження;

– метрологічні характеристики;

– алгоритмічне та програмне забезпечення;

8) висновки;

9) література;

10) глосарій;

11) додатки;

II) носій інформації на якому розміщені текстова частина роботи, програми, матеріали та презентація доповіді.

Наведені заголовки основної частини є рекомендованими для КР(МР). Основна частина КР(МР) повинна мати аналітично-розрахунковий характер.

Дозволяється вводити нові частини за вказівками керівника.

Кожний розділ ТЧ може складатися з підрозділів, пунктів, підпунктів, параграфів.

Обсяг ТЧ повинен складати 30-50 сторінок машинописного тексту на аркушах фор­мату А4, причому об’єм основної частини повинен складати не мен­ше 70 % всієї роботи.

**Вступ**

В сучасний період розвитку обчислювальної техніки актуальність числових методів, що дозволяють розв’язувати широкий клас задач за допомогою ЕОМ, продовжує зростати. Особливо гостро постає питання оптимального використання усіх запропонованих компонентів з архітектури комп’ютера. За останні 10 років розвиток графічних процесорів (GPU) при використанні у сукупності із центральним процесором (CPU) нестримними темпами відкриває все нові можливості ЕОМ. З одного боку, як і раніше продовжується приріст продуктивності ЕОМ за рахунок збільшення кількості процесорів. З іншого боку, гібридні системі стають більш популярні, що зумовлено використанням елементів принципово нової архітектури.

**Актуальність роботи** зумовлена нестримним рухом технологій з плином часу. Таким чином на вже розв’язані задачі можна подивитися під іншим кутом, а саме використання гібридних комп’ютерів для розв’язання диференціальних рівнянь.

Використання комп’ютера встановлює задачу раціонального використання ресурсів, адже навіть в наш час ресурсами є скінчені величини, якими не можна розкидатись. Зараз ми маємо багатопроцесорні системи, які можуть виконувати декілька процесів одночасно, або майже одночасно. Проте архітектура ЦПУ достатньо обмежена, тоді як ГПУ дає можливості дуже великої кількості процесорів, якої позбавлений ЦПУ. А саме виконання операцій одночасно у великих кількостях. Звичайно використання лише однієї частини комп’ютера гібридної архітектури не може дати оптимального часу, тож є сенс для розв’язання такої задачі використовувати обидва процесора.

Розвиває та покращує ринок ГПУ на сьогодні компанія NVidia та AMD. Її розробка CUDA (Compute Unified Device Architecture) або ATI Stream Technology дає можливості використовувати можливості GPU для запуска обчислювальних програм. Різниця між використанням багатопроцесорного одного керуючого пристрою та гібридної архітектури полягає у наступному:

Перший потребує від алгоритмів більшої степені паралелізма на однотипових процесорних ядрах, які на програмному рівні вирішуються за допомогою спеціальних програмних систем такі як MPI. Другий потребує від алгоритма більш складної багаторівневої паралельної моделі. Такі системи потребують відповідей на додаткові запитання до алгоритма та використовування пам’яті.

Розв’язання диференціальних рівнянь завжди було суттєвою проблемою багатьох задач з моменту існування таких задач. Розв’язуючи ту чи іншу реальну проблему за допомогою математики дослідники будують математичні моделі, які в свою чергу у багатьох випадках зводяться до розв’язання диференціального рівняння бо саме диференційні рівняння краще за будь які інші окреслюють суть процесу.

Було б чудово, якщо ЕОМ мали засоби вирішувати такий великий клас задач аналітично, проте саме лише використання машини для пошуку розв’язка змушує нас відмовитися від точних розв’язків на користь наближених в деякому наборі точок. Замість точної задачі можна використати наближення диференційного оператора у вигляді різницевої схеми, поставити йому у відповідність граничні початкові умови і знайти наближений розв’зок за допомогою системи лінійних алгебраїчних рівнянь.

Використання таких систем і залежність від їх розв’язку має не лише клас диференційних рівнянь, а й багато областей науки й підприємств. Не зважаючи на велику увагу до створення програмного забезпечення з лінійної алгебри багато проблем ефективного його використання залишаються.

Такі машинні алгоритми здебільшого передбачають розв’язання задач із потрібною точністю та будь якою кількістю потрібних точок, де ми можемо знайти наближений розв’язок. Список методів для розв’язання яких можемо перераховувати достатньо довго (метод Гауса, метод Гауса-Жордана, метод Гауса-Зейделя, матричний метод розв’язання систем лінійних алгебраїчних рівнянь, метод квадратного кореня, метод Крамера, метод прогонки, метод Якобі, метод релаксації, розгалуження Холецького, проекційні методи, метод регуляризації Тихонова, ітераційні методи, метод Річардсона).

Обирання того чи іншого метода залежить від вигляду отриманої системи, її властивостей, бажаної швидкості знаходження розв’язку. Для використання методу на ЕОМ, маючи різницеву залежність для знаходження наближеного розв’язку, ітеративні матимуть перевагу при розпаралелюванні. Кожна наступна ітерація знаходиться з використанням попередніх обчислень, доки процес не збіжиться в деякій точці. Гарантія збіжності забезпечується кроками методу та початковим наближенням. Такі методи, маючи коректну постановку збігаються при достатній кількості операцій. Тож головне питання для ітеративного процесу складається у швидкості знаходження наближення, бо в деяких умовах використання методів, як для швидкого обчислення у реальному часі, при надточних обчисленнях можуть залежати не тільки отримані розв’язки, приймання рішення, а навіть життя людини.

**Метою роботи** є отримання результату оптимізації за допомогою CUDA та надання методів паралельного обчислювання таких задач.

**Основна частина**

аналіз існуючих методів (алгоритмів)розв’язання поставленої задачі;

На сучасному ринку можна виокремити 2 конкурентні програмно-апаратні архітектури, за допомогою яких можливо використати повну потужність гібридного комп’ютера

NVidia CUDA

ATI Stream Technology

В даній роботі була обрана CUDA як досліджувана технологія, з наступних причин:

* Інтерфейс програмування додатків CUDA (CUDA API) заснований на стандартній мові програмування С з деякими обмеженнями. За думкою розробників це повинно спростити та пом’якшити процес вивчення архітектури CUDA.
* Поділена між потокам пам’ять (shared memory) розміром у 16 Кб може бути використана під організований користувачем кеш з більш ширшою полосою пропуску ніж при виборці зі звичайних текстур.
* Більш ефективні транзакції між пам’яттю центрального процесора та відеопам’яттю.
* Повна апаратна підтримка цілочисельних та бітових операцій.
* Підтримка компіляції GPU кода коштами відкритого LLVM (низькорівнева віртуальна машина).

З обмежень маємо

Усі функції, виконані на пристрої не підтримують рекурсії.

Такі висновки представлені при порівнянні CUDA з традиційним підходом до організації обчислювань загального призначення за допомогою можливостей графічних API.

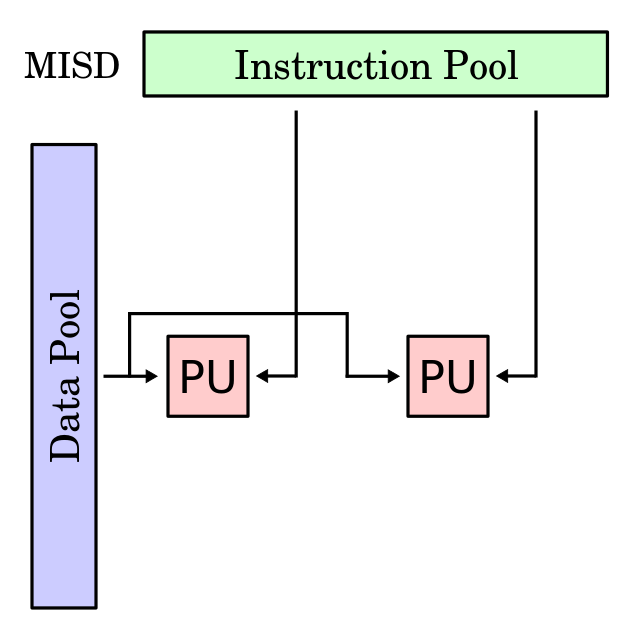
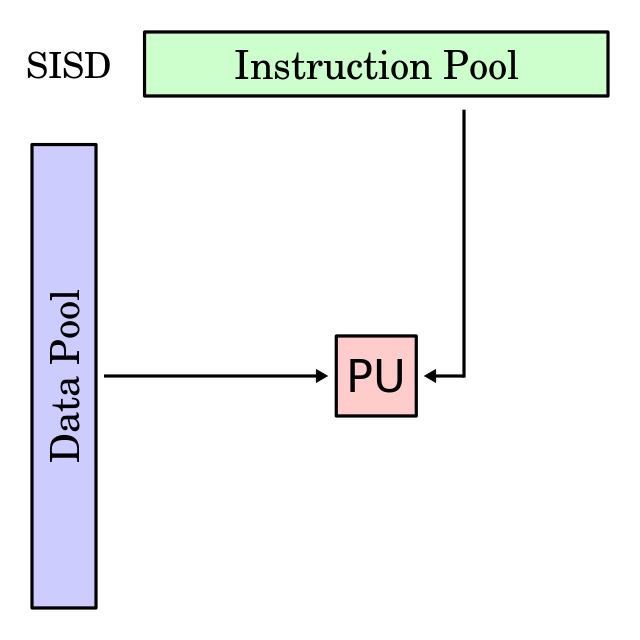
За класифікацією по Фліну архітектура може мати наступні варианти реалізації:

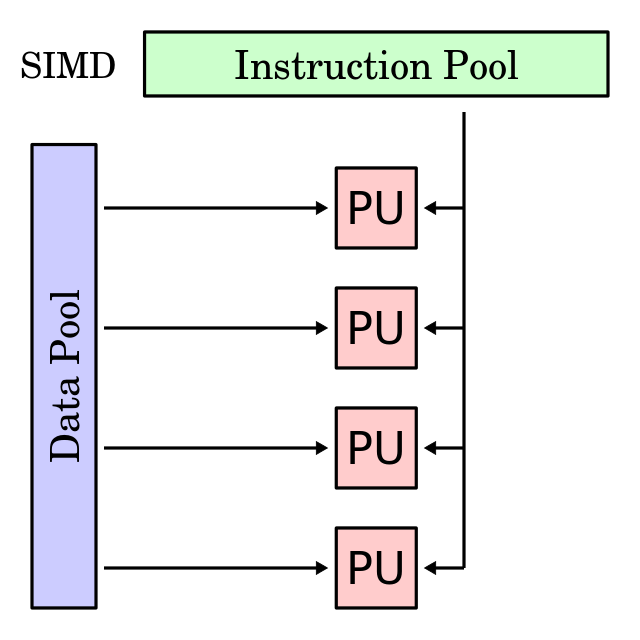
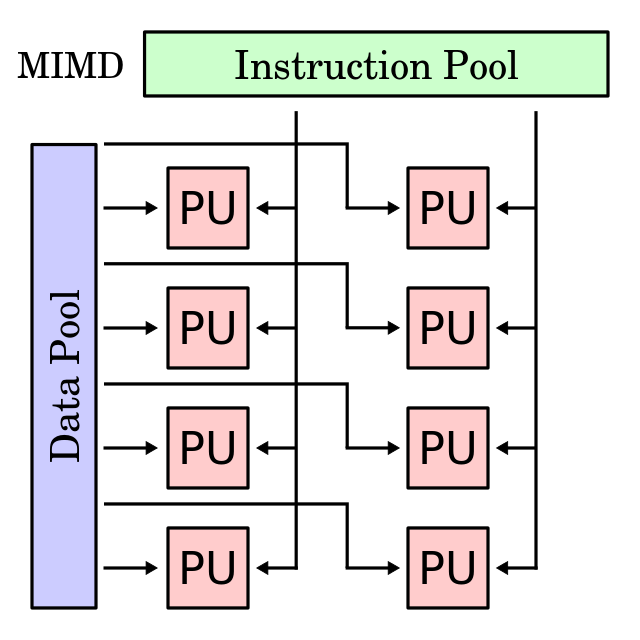
* SISD
* MISD
* SIMD
* MIMD

(Single/Multiple Instruction Single/Multiple Data)

За цією класифікацією ЕОМ поділяються на 4 типа за кількістю потоків команд та даних. За звичай попередники сучасних комп’ютерів мали одну головну архітектуру SISD. Паралельно йшла розробка векторних та матричних архітектур MISD, SIMD та багатопроцесорні MIMD. Кожна архітектура має свої особливості та свої як гарні так і погані сторони. Методи розпаралелювання можуть бути оптимізовані для одних чи інших типів архітектур.

Вигляди архітектур:

Складності, які виникають при розробці такого забезпечення мають такі характери:

* Програма повинна бути складена з кода для CPU (на звичайній мові програмування С / С++) та кода для графічного процесора написаного на спеціальній мові, CUDA.
* Друга проблема пов’язана з ефективним використанням обчислювальних ресурсів, з узгодженням розподілу обчислювальних ресурсів на ядрах (GPU та CPU).

Щоб дати відповідь на запитання чи має сенс використовувати той чи інший метод для даних архітектур важливо мати коефіцієнти, що характеризують ефективність паралельних алгоритмів. Розпаралелювання обчислень багатоваріантно, тобто для MIMD машин з однією й тою ж самою структурою міжпроцесорних зв’язків можуть бути побудовані різноманітні варіанти алгоритмів про паралельну організацію обчислень. Для порівняння й оцінки якості алгоритмів паралельних обчислень будемо користуватися такими критеріями, як коефіцієнт прискорення та коефіцієнт ефективності:

Деякі автори вводять й інші характеристики:

У подальшому передбачається, що необхідна для реалізації обчислювального алгоритма інформація зберігається та обробляється в оперативній пам’яті ЕОМ або ж у сумарній пам’яті MIMD машини, тобто усюди передбачається не використання зовнішньої пам’яті.

Опис задачі:

У якості модельної задачі розглядаємо задачу Діріхлє для самоспряжених рівнянь другого порядку в прямокутнику

На сітці поставимо у відповідність різницеву задачу

Для запису системи лінійних алгебраїчних рівнянь в матрично-векторному вигляді необхідно спочатку встановити відповідність між впорядкуванням рівнянь та впорядкуванням невідомих. Роздивимось два впорядкування невідомих. Природне впорядкування та червоно-чорне.

На даному етапі можна вже описати алгоритм дій програми для першого методу Річардсона, або явного чебишевського методу ще відомого як ітераційний метод з Чебишевським набором параметрів.

1. Підготовка та налаштування глобальних змінних
   1. Кількість вузлів
   2. Допустима похибка
   3. Початок часу виконання
   4. Налаштування різницевої матриці
   5. Права частина на сітці
   6. Перше наближення
2. Розрахунок додаткових змінних
   1. Розрахування спектру матриці
   2. Масив оптимального чебишевського набору кроків
   3. Знаходження мінімальної потрібної кількості операцій
3. Обробка головного циклу
   1. Проведення ітерацій по наближенню за формулами
4. Вивід та аналіз результатів

Аналогічний алгоритм використовується й для обробки програми на CUDA. Лише функція головного циклу оптимізується на GPU.

Отримані результати свідчать, що на маленьких системах використання GPU не є раціональною ідеєю, бо саме використання такого пристрою затрачує деякий час, але при зростанні кількості операцій, складності системи оптимізація проявляється і оптимізований алгоритм виконується в декілька разів швидше.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Розмір сітки | 5 | 10 | 25 | 50 |
| Ітерації | 50 | 220 | 1375 | 5500 |
| C++ (сек.) | 0.012 | 0.05 | 5.88 | 95 |
| CUDA (сек.) | 2.27 | 2.28 | 4.77 | 40.5 |

Таким чином вже на цій стадії роботи можна відмітити пропорційність росту часу в одному та іншому випадках і сказати, що оптимізація при використанні гібридних архітектур помітна, і саме за такими обчисленнями може стати майбутнє обчислювальної математики.