МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ВОЛОДИМИРА ДАЛЯ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

“ТЕХНОЛОГІЇ РОЗПОДІЛЕНИХ СИСТЕМ ТА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ"

(для здобувачів вищої освіти напрямку 6.050101 «Комп'ютерні науки» та «Компютерна інженерія»)

(Електронне видання)

ЗАТВЕРДЖЕНО
на засіданні кафедри
комп'ютерних наук та інженерії
протокол №2 від 04.10.2021 р.
04.10.2021р.

Сєвєродонецьк 2021 р.

УДК 681.3.013

Методичні вказівки до лабораторних робіт з дисципліни "Технології розподілених систем та паралельних обчислень" (для здобувачів вищої освіти напрямків «Комп'ютерна інженерія» та «Комп'ютерні науки») /Уклад.: Д.О. Недзельський. – Сєвєродонецьк: вид-во СНУ ім. В. Даля, 2021. – 15 с.

Методичне видання спрямоване на вивчення та засвоєння здобувачами вищої освіти теоретичних основ та практичного матеріалу з дисципліни **"**Технології розподілених систем та паралельних обчислень**"**. Методичні вказівки містять теоретичні матеріали для виконання робіт; описи алгоритмів; приклади програм; завдання для виконання лабораторних робіт; вимоги до оформлення звітів; контрольні питання для підготовки до захисту.

Методичні матеоіалі розраховані на здобувачів вищої освіти закладів вищої освіти.

Укладач Д.О. Недзельський, к.т.н., доц.

Рецензент В.С. Кардашук, к.т.н., доц.

ЗМІСТ

1. ВСТУП

Метою лабораторних робіт є поглиблення теоретичних знань по дослідженню ефективності виконання програми множення матриць та рішення диференціальних рівнянь з приватними похідними в багатоядерних системах з спільною оперативною

Об' єктом дослідження є програми.

2. ЛАБОРАТОРНА РОБОТА №1

Тема. Дослідження ефективності виконання програми множення матриць в багатоядерній системі з спільною оперативною пам'яттю.

Мета роботи: Дослідження ефективності виконання програми множення матриць в багатоядерних системах з спільною оперативною пам'яттю.

2.1Постановка завдання

Перемноження матриці A m × n розміру і матриці B n ×l розміру призводить до отримання матриці C m × l розміру, кожен елемент якої визначається відповідно до виразу:

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}, 0 \le i < m, 0 \le j < l .$$

Очевидно, що кожний елемент матриці C є скалярний добуток відповідних рядка матриці A і стовпця матриці B:

$$c_{ij} = a_i, b_j^T \ , \ a_i = a_{i0}, a_{i1}, \ldots, a_{in-1} \ , \ b_j^T = b_{0j}, b_{1j}, \ldots, b_{n-1j}^{\ T} .$$

Цей алгоритм передбачає виконання m * n * l операцій множення і стільки ж операцій додавання елементів матриць. При множенні квадратних матриць розміру n × n кількість виконаних операцій має порядок $O(n^3)$.

2.2 Послідовний алгоритм

Опис алгоритму

Послідовний алгоритм множення матриць характеризується трьома вкладеними циклами:

```
// Програма 1
// Послідовний алгоритм множення матриць
double MatrixA [Size] [Size];
double MatrixB [Size] [Size];
double MatrixC [Size] [Size];
int i, j, k;
...
```

```
for (i = 0; i <Size; i ++) {
 for (j = 0; j <Size; j ++) {
 MatrixC [i] [j] = 0;
 for (k = 0; k <Size; k ++) {
 MatrixC [i] [j] = MatrixC [i] [j] + MatrixA [i] [k] * MatrixB [k] [j];
 }
 }
}
```

Цей алгоритм є ітеративним і орієнтований на послідовне обчислення рядків матриці C . Дійсно, при виконанні однієї ітерації зовнішнього циклу (циклу по змінній i) отримується один рядок матриці C (рис. 1).

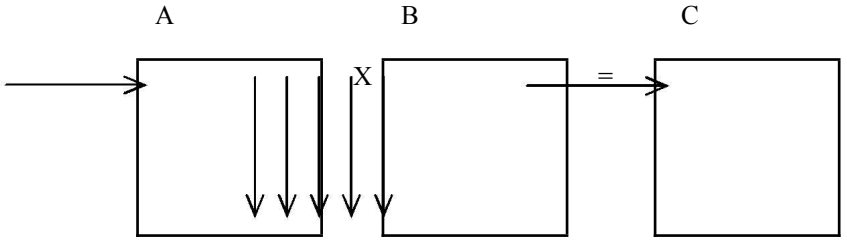

Рис. 1 - На першій ітерації циклу по змінній i використовується перший рядок матриці A і всі стовпці матриці B для того, щоб вичислити елементи першого рядка матриці C

Можливий варіант послідовної програми множення матриць.

1. Головна функція програми. Реалізує логіку роботи алгоритму, послідовно викликає необхідні підпрограми.

```
1        Програма 2
2        Послідовний алгоритм множення матриць void main (int argc, char *
argv []) {
         double * pAMatrix;        // Матриця A
         double * pBMatrix;        // Матриця B
         double * pCMatrix;        //   матриця C
         int Size;                 //   Розмір матриць

       // Ініціалізація даних
          ProcessInitialization (pAMatrix, pBMatrix, pCMatrix, Size);
       // Множення матриць
          SerialResultCalculation (pAMatrix, pBMatrix, pCMatrix, Size);
       // Завершення обчислень
          ProcessTermination (pAMatrix, pBMatrix, pCMatrix, Size);
}
```

2. Функція ProcessInitialization. Ця функція визначає розмір матриць і елементи для матриць A і B, результуюча матриця C заповнюється нулями. Значення елементів для матриць A і B визначаються у функції RandomDataInitialization.

```
// Функція виділення оперативної пам'яті та  ініціалізації даних void
ProcessInitialization (double * & pAMatrix,
    double * & pBMatrix, double * & pCMatrix, int & Size)
{
    int i, j;
    do {
       printf ( "\ nВведіте розмір матриць:"); scanf ( "% d", & Size);
       if (Size <= 0) {
          printf ( "Розмір має бути більше 0! \ n");
```

```
        }
      } While (Size <= 0);
      pAMatrix = new double [Size * Size]; pBMatrix = new double
[Size * Size]; pCMatrix = new double [Size * Size]; for (i = 0; i <Size; i
++)
            for (j = 0; j <Size; j ++) pCMatrix [i * Size + j] =
 0;
RandomDataInitialization (pAMatrix, pBMatrix, Size);
}
```

Реалізація функції RandomDataInitialization пропонується для самостійного виконання. Вихідні дані можуть бути введені з клавіатури, прочитані з файлу або згенеровані за допомогою датчика випадкових чисел.

3. Функція SerialResultCaculation. Ця функція виконує множення матриць.

```
// Функція матричного множення
   void SerialResultCalculation (double * pAMatrix, double * pBMatrix, double *
pCMatrix, int Size) {int i, j, k;
        for (i = 0; i <Size; i ++) for (j = 0; j <Size; j ++)
          for (k = 0; k <Size; k ++)
              pCMatrix [i * Size + j] + = pAMatrix [i * Size + k] * pBMatrix [k *
 Size + j];
}
```

Слід зазначити, що наведена програма може бути оптимізована (обчислення індексів, використання кеш-пам'яті і т. п.).

2.3 Базовий паралельний алгоритм множення матриць

Розглянемо паралельний алгоритм множення матриць, в основу якого покладено розбиття матриці A на послідовності рядків (горизонтальні смуги).

Цей алгоритм орієнтований на обчислення рядків матриці C. Дійсно, при множенні одного рядка матриці A на всю матрицю B отримується один рядок матриці C. (рис. 2).

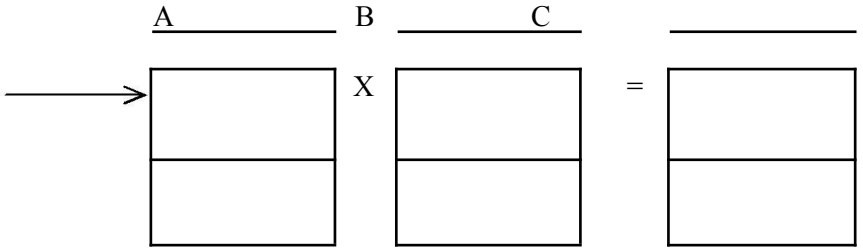

Рис. 2

2.4 Визначення підзадач

З визначення операції матричного множення витікає, що обчислення всіх елементів матриці C може бути виконано незалежно одне від одного. Як результат, можливий підхід для організації паралельних обчислень полягає у використанні в якості базової підзадачі процедури визначення одного елемента матриці C. Для проведення всіх необхідних обчислень кожна підзадача повинна викнувати обчислення над елементами одного рядка матриці A та одного стовпця матриці B. Загальна кількість одержуваних при такому підході підзадач виявляється рівним n^2 (по числу елементів матриці C).

Розглянувши запропонований підхід, можна відзначити, що досягнутий рівень паралелізму є в деякій мірі надлишковим. При проведенні практичних розрахунків кількість сформованих підзадач, як правило, буде перевищувати число наявних обчислювальних елементів (ядер процесорів) і, тим самим, неминучим є етап укрупнення базових задач. В цьому плані може виявитися корисним агрегація обчислень вже на етапі виділення базових підзадач. Можливе рішення може полягати в об'єднанні в рамках однієї підзадачі всіх обчислень, пов'язаних не з одним, а з декількома елементами результуючої матриці C. Для подальшого розгляду визначимо базову задачу як процедуру обчислення всіх елементів одного з рядків матриці C. Такий підхід призводить до зниження загальної кількості підзадач до величини **n**.

Для виконання всіх необхідних обчислень базової підзадачі повинні бути доступні один з рядків матриці A і всі стовпці матриці B. Просте рішення цієї проблеми - дублювання матриці B у всіх підзадач. Слід зазначити, що такий підхід не призводить до реального дублювання даних, оскільки цей алгоритм орієнтований на застосування для обчислювальних систем із спільною оперативною пам'яттю, до якої є доступ з усіх використовуваних обчислювальних елементів.

2.5 Виділення інформаційних залежностей

Для обчислення одного рядка матриці C необхідно, щоб в кожній підзадачі містився рядок матриці A і був забезпечений доступ до всіх стовпців матриці B. Спосіб організації паралельних обчислень представлений на рис. 3.

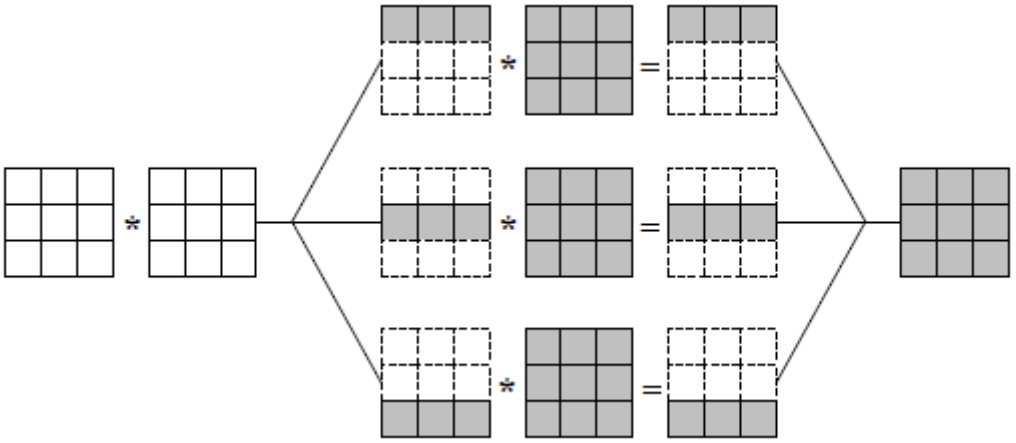

Рис. 3 - Організація обчислень при виконанні паралельного алгоритму множення матриць, заснованого на поділі матриць по рядках

2.6 Масштабування і розподіл підзадач

Виділені базові підзадачі характеризуються однаковою обчислювальної трудомісткістю і рівним обсягом переданих даних. В разі, коли розмір **n** матриць виявляється більшим, ніж число **p** обчислювальних елементів (ядер в процесорах), базові підзадачі можна укрупнити, об'єднавши в рамках однієї підзадачі кілька сусідніх рядків матриці. В цьому випадку вихідна матриця A і матриця-результат C розбиваються на ряд горизонтальних смуг. Розмір смуг при цьому слід вибрати рівним **k = n / p** (в припущенні, що **n** кратне **p**), що дозволить як і раніше забезпечити рівномірність розподілу обчислювального навантаження по обчислювальним елементам (ядрам процесорів).

2.7 Програмна реалізація паралельного (багатопотокового) варіанта

Для того, щоб розробити паралельну програму, яка реалізовує описаний підхід за допомогою технології OpenMP, необхідно в послідовний (однопотоковий) варіант програми додати одну директиву parallel for в функції SerialResultCaclualtion (назвемо новий варіант функції ParallelResultCaclualtion):

Програма 3

Функція паралельного матричного множення void ParallelResultCalculation (double * pAMatrix,

double * pBMatrix, double * pCMatrix, int Size) {int i, j, k;

#pragma omp paralell for private (j, k) for (i = 0; i <Size; i ++)

for (j = 0; j <Size; j ++) for (k = 0; k <Size; k ++)

pCMatrix [i * Size + j] + = pAMatrix [i * Size + k] * pBMatrix [k * Size + j];

}

Ця функція виконує множення рядків матриці A на стовпці матриці B з використанням декількох паралельних потоків. Кожен потік виконує обчислення над декількома сусідніми рядками матриці A і, таким чином, отримує кілька сусідніх рядків результуючої матриці C.

2.8 Завдання для лабораторної роботи №1

Вивчити технологію програмування Open MP.
Завдання у вигляді кількості ядер і параметрів системи видає викладач.
Розробити послідовний та багатопотоковий варіанти програми множення матриць.
Провести ряд експериментів з розробленими програмами.
Результати експериментів представити в таблиці.

Таблиця - Результати обчислювальних експериментів для паралельного алгоритму множення матриць при стрічковій схемі поділу даних по рядках

Розмір матриці	Послідовний Алгоритм (час)	паралельний алгоритм					
		2 потоки		4 потоки		8 потоків	
		Час	Прискорення	Час	Прискорення	Час	Прискорення
1000							
2000							
3000							
4000							
5000							

Зробити висновки про ефективність паралельного виконання програми.

2.9 Зміст звіту по лабораторній роботі №1

Звіт має бути оформлений на листах формату А4.
Загальний зміст звіту:
формулювання завдання;

короткі відомості про особливості технології програмування Open MP;
однопотоковий варіант програми;
багатопотоковий варіант програми;
результати експериментів;
висновки.

2.10 Контрольні питання

1. Які підходи використовуються для розробки паралельних програм?
2. В чому полягають основи технології OpenMP?
3. В чому полягають основні переваги технології OpenMP?
4. Що розуміється під паралельною програмою в рамках технології OpenMP?
5. Що розуміється під поняттям потоку (thread)?
6. Які проблеми виникають при використанні спільних даних в паралельно виконуваних потоках?
7. Який формат запису директив технології OpenMP?
8. Якій структурі комп'ютера краще відповідає технологія Open MP - кластеру чи багатоядерній системі з спільною оперативною пам'яттю?
9. Чим відрізняються поняття "процес" і "потік"?
10. Як розподіляються дані в розробленій програмі?
11. Як визначаються підзадачі?
12. Як виділяються інформаційні залежності?
13. Як розподіляються підзадачі?
14. Чим розрізняються спільні та локальні змінні в паралельній секції Open MP-програми?

3 ЛАБОРАТОРНА РОБОТА №2

Тема. Дослідження ефективності виконання програми рішення диференціальних рівнянь з приватними похідними в багатоядерній системі з спільною оперативною пам'яттю.

Мета роботи – дослідження ефективності виконання програми рішення диференціальних рівнянь з приватними похідними в багатоядерній системі з спільною оперативною пам'яттю.

3.1Вступ

Диференціальні рівняння в приватних похідних э широко застосовуваним математичним апаратот при розробці моделей в самих різних областях науки і техніки. На жаль, явне рішення цих рівнянь в аналітичному вигляді виявляється можливим тільки в досить простих випадках, тому можливість аналізу математичних моделей, побудованих на основі диференціальних рівнянь, забезпечується за допомогою наближених чисельних методів рішення. Обсяг виконуваних при цьому обчислень зазвичай є значним і використання високопродуктивних обчислювальних систем є традиційним для даної області обчислювальної математики.

Розглянемо проблему чисельного рішення задачі Діріхле для рівняння Пуассона, яка визначається як задача знаходження функції u=u (xy), що задовольняє в області визначення D рівняння

$$\begin{cases} \dfrac{\delta^2 u}{\delta x^2}+\dfrac{\delta^2 u}{\delta y^2}=f(x,y), & (x,y)\in D, \\ u(x,y)=g(x,y), & (x,y)\in D^0, \end{cases}$$

і приймає значення g(x,y) на кордоні D0 області (і є функціями, які задаються при постановці задачі). Подібна модель може бути використана для опису сталого плину рідини, стаціонарних теплових полів, процесів теплопередачі з внутрішніми джерелами тепла і деформації пружних пластин та інші. Даний приклад часто використовується в якості навчально-практичного завдання при викладі можливих способів організації ефективних паралельних обчислень.

Для простоти викладу матеріалу в якості області завдання D0 функції g (x, y) далі буде використовуватися одиничний квадрат.

3.2 Послідовні методи розв'язання задачі Діріхле

Одним з найбільш поширених підходів чисельного рішення диференціальних рівнянь є метод кінцевих різниць (метод сіток). Слідуючи цьому підходу, область вирішення D представляється у вигляді дискретного (як правило, рівномірного) набору (сітки) точок (вузлів). Так, наприклад, прямокутна сітка в області D може бути задана у вигляді

$$\begin{cases} D_h=\{(x_i,y_j): x_i=ih,\ y_i=jh,\ 0\le i,j\le N+1, \\ \qquad\qquad h=1/(N+1), \end{cases}$$

де величина N задає кількість вузлів по кожній з координат області D.

Позначимо оцінювану при подібному дискретному поданні апроксимацію функції u (xy) в точках (xiyj) через **u**ij. Тоді, використовуючи п'ятиточковий шаблон для обчислення значень похідних, можна представити рівняння Пуассона в кінцево-різницевій формі таким чином

$$\frac{u_{i-1,j}+u_{i+1,j}+u_{i,j-1}+u_{i,j+1}-4u_{ij}}{h^2}=f_{ij}.$$

Дане рівняння може бути розв'язано щодо **u**ij:

$$u_{ij}=0.25(u_{i-1,j}+u_{i+1,j}+u_{i,j-1}+u_{i,j+1}-h^2 f_{ij}).$$

за яким чергове k-е наближення значення ij **u** обчислюється за останнім k-м наближенням значень **u**i-1,j та **u**i,j-1 і передостаннього (k-1)-го наближення значень **u**i-1,j та **u**i,j-1. Виконання ітерацій зазвичай триває до тих пір, поки одержувані в результаті ітерацій зміни значень **u**ij не стануть менше деякої заданої величини (необхідної точності обчислень). Збіжність описаної процедури (отримання рішення з будь-якою бажаною точністю) є предметом всебічного математичного аналізу, тут же відзначимо, що послідовність рішень, одержуваних методом сіток, рівномірно сходиться до вирішення задачі Діріхле, а похибка рішення має порядок 2h.

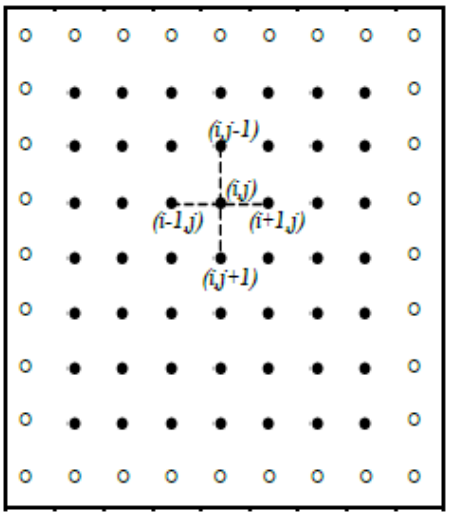

Рис.1 - Прямокутна сітка в області D (темні точки представляють внутрішні вузли сітки, нумерація вузлів в рядках зліва направо, а в стовпчиках - зверху вниз)

3.3 Розробка послідовного варіанта програми

Варіант послідовної програми з використанням методу сіток може бути отриманий, якщо дозволити довільний порядок перерахунку значень. Програма для даного способу обчислень може бути представлена в наступному вигляді:

```
for (I = 1; i <n + 1; i ++) {
        temp = u [i] [j];
                u [i] [j] = 0.25 * (u [i-1] [j] + u [i + 1] [j] + u [i] [j-1] + u [i] [j + 1] -h * h *
f [i] [j]);
                d = fabs (temp - u [i] [j]);
                dmax + = d;
        }
```

```
            // omp_set_lock (& dmax_lock);
           //          if (dmax <dm) dmax = dm;
// omp_unset_lock (& dmax_lock);
}
dmax / = n * n; // визначення середньої похибки
if (Dmax> eps) kPogr = l + 1;
```

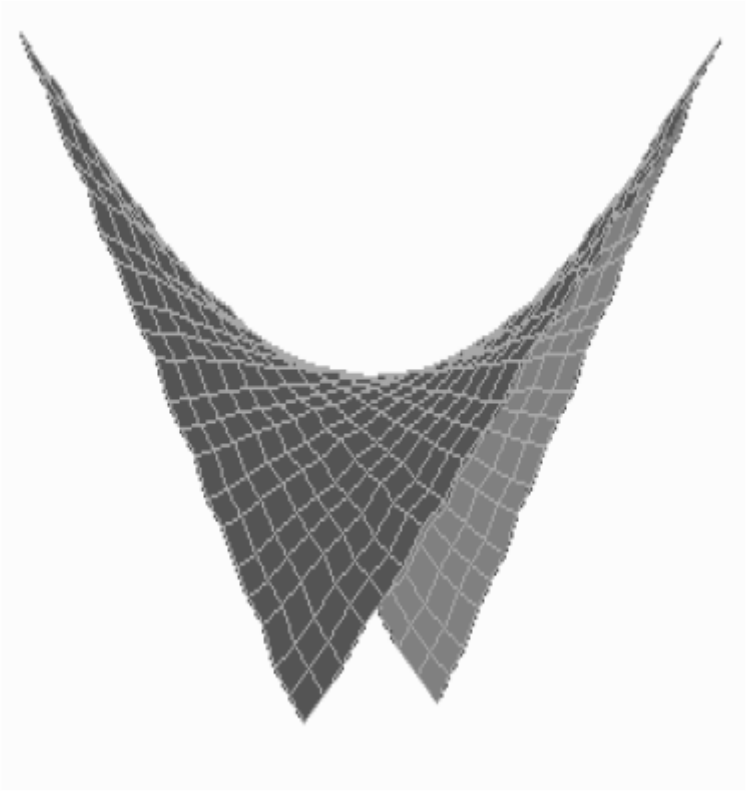

Рис. 2 - f (x,y) = x * x + y * y (сума квадратів)

3.4 Розробка паралельного варіанта програми

Варіант паралельної програми при використанні технології OpenMP отримано, при використанні довільного порядку перерахунку значень. Програма для даного способу обчислень може бути представлена в наступному вигляді:

```
omp_lock_t dmax_lock;
omp_init_lock (& dmax_lock);
omp_set_num_threads (2); // явна установка кількості потоків
int kPogr = 0;
start = clock ();
do {
dmax = 0;
tCore = 0;
#pragma omp parallel for shared (u, n, dmax, k) private (I, j, temp, d, dm, tCore)
for (J = 1; j <n + 1; j ++) {
// core1 = clock ();
dm = 0;
for (I = 1; i <n + 1; i ++) {
temp = u [i] [j];
u [i] [j] = 0.25 * (u [i-1] [j] + u [i + 1] [j] + u [i] [j-1] + u [i] [j + 1] -h * h*f [i] [j]);
```

```
d = fabs (temp - u [i] [j]);
dmax + = d;
}
omp_set_lock (& dmax_lock);
if (Dmax <dm) dmax = dm;
omp_unset_lock (& dmax_lock);
// core2 = clock ();
// tCore + = (double) (core2-core1);
}

dmax / = n * n; // визначення середньої похибки
if (Dmax> eps) kPogr = K - k + 1;
k--;
} while (K> 0);
```

Слід зазначити, що програма отримана з вихідної послідовної програми шляхом додавання директив і операторів звернення до функцій бібліотеки OpenMP.

Як випливає з тексту програми, паралельні області в даному прикладі задаються директивою parallel for, є вкладеними і включають в свій склад оператори циклу for. Компілятор, який підтримує технологію OpenMP, розділяє виконання ітерацій циклу між декількома потоками програми, кількість яких зазвичай збігається з числом ядер в обчислювальній системі.

Параметри директиви shared та private визначають доступність даних в потоках програми - змінні, описані як shared, є спільними для потоків, для змінних з описом private створюються окремі копії для кожного потоку, які можуть використовуватися в потоках незалежно один від одного.

Наявність спільних даних забезпечує можливість взаємодії потоків. В цьому плані спільні змінні розглядаються як загальні ресурси потоків і, як результат, їх використання виконується з дотриманням правил взаємовиключення (зміна будь-яким потоком значень спільних змінних повинна призводити до блокування доступу до модифікуємих даних для всіх інших потоків). В даному прикладі таким ресурсом є змінна dmax, доступ потоків до якої регулюється спеціальною службовою змінною (семафором) dmax_lock і функціями omp_set_lock (дозвіл або блокування доступу) та omp_unset_lock (зняття заборони на доступ). Подібна організація програми гарантує єдиність доступу потоків для зміни спільних даних.

Ділянки програмного коду (блоки між зверненнями до функцій omp_set_lock і omp_unset_lock), для яких забезпечується взаємовиключення, це критичні секції.

Розроблений паралельний алгоритм є коректним, який забезпечує рішення поставленого завдання. Використаний при розробці підхід забезпечує досягнення практично максимально можливого паралелізму. Проте, результат не може бути визнаний задовільним - програма буде працювати повільно і замість очікуваного прискорення обчислень від використання декількох ядер отримаємо уповільнення розрахунків.

Шлях для досягнення ефективності паралельних обчислень лежить в зменшенні необхідних моментів синхронізації паралельних ділянок програми. Так, в розглянутому прикладі можна обмежитися розпаралелюванням тільки одного зовнішнього циклу for. Крім того, для зниження кількості можливих блокувань можна застосувати для оцінки максимальної похибки багаторівневу схему розрахунку: нехай паралельно виконується потік, який спочатку формує локальну оцінку похибки dm тільки для своїх оброблюваних даних (однієї або декількох рядків сітки), а при завершенні обчислень потік порівнює свою оцінку dm з загальною оцінкою похибки dmax.

3.5 Завдання для лабораторної роботи №2

Вивчити:

технологію програмування Open MP;

послідовний та паралельний алгоритми рішення диференціальних рівнянь в приватних похідних.

Розробити послідовний та багатопотоковий варіанти програми рішення диференціальних рівнянь в приватних похідних;.

Провести ряд експериментів з розробленими програмами.

Результати експериментів представити в таблиці.

Таблиця - Результати обчислювальних експериментів, виконаних для оцінки ефективності послідовного і паралельного варіантів програми рішення диференціальних рівнянь з приватними похідними (використаний алгоритм Гаусса-Зейделя при 4 ядрах) Кількість ітерацій 200, час в сек.

Розмір сітки	Послідовний Алгоритм (час)	паралельний алгоритм					
		2 потоки		4 потоки		8 потоків	
		Час	Прискорення	Час	Прискорення	Час	Прискорення
100	200						
500	200						
1000	200						
2000	200						
4000	200						

Зробити висновки про ефективність паралельного виконання програми.

3.6 Зміст звіту по лабораторній роботі №2

Звіт має бути оформлень на листах формату А4.

Загальний зміст звіту:

формулювання завдання;

короткі відомості про методи рішення диференціальних рівнянь в приватних похідних

послідовний та паралельний алгоритми рішення диференціальних рівнянь в приватних похідних;

основні особливості технології паралельного програмування Open MP;

тексти розроблених послідовної та паралельної програм;

результати експериментів;

висновки.

3.7 Контрольні питання

1. Які типи паралельних систем вам відомі?
2. В чому полягають основи технології OpenMP?

3. В чому полягають основні переваги технології OpenMP?
4. Що розуміється під паралельною програмою в рамках технології OpenMP?
5. Що розуміється під поняттям потоку (thread)?
6. Які проблеми виникають при використанні спільних даних в паралельно виконуваних потоках?
7. Які методи можуть бути використані для вирішення диференціальних рівнянь в приватних похідних?
8. В чому ідея паралельної реалізації диференціальних рівнянь в приватних похідних?
9. Які інформаційні взаємодії є між базовими підзадачами для паралельного варіанта?
10. Які підходи використовуються для розробки паралельних програм?
11. Якій структурі комп'ютера краще відповідає технологія Open MP - кластеру чи багатоядерній системі з спільною оперативною пам'яттю?
12. Чим відрізняються поняття "процес" і "потік"?
13. Як розподіляються дані в розробленій програмі?
14. Як визначаються підзадачі?
15. Як виділяються інформаційні залежності?
16. Як розподіляються підзадачі?
17. Чим розрізняються спільні та локальні змінні в паралельній секції Open MP-програми?
18. Які показники ефективності для паралельного варіанта?

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

“Технології розподілених систем та паралельних обчислень"

(для здобувачів вищої освіти напрямку 6.050101 «Комп'ютерні науки» та "Комп'ютерна інженерія")

(Електронне видання)

Редактор - Д.О. Недзельський

Техн. редактор - Д.О. Недзельський

Оригінал - Д.О. Недзельський

Підписано до друку _____
Формат 60×84$^{1}/_{16}$ Папір типограф. Гарнітура Times.
Друк офсетний. Умов. друк. арк. 2. Обл.–вид. арк. _____.
Тираж __ прим. Вид. № _____. Замов № _____. Ціна договірна.

Видавництво Східноукраїнського національного університету
імені Володимира Даля

Адреса видавництва: 93406, м. Сєвєродонецьк, Луганської обл.,
пр. Центральний, 59–а, головний корпус
Телефон: (06452) 4–03–42 E–mail: vidavnictvosnu.ua@gmail.com