Змістовний модуль:

РОЗРОБКА ПАРАЛЕЛЬНИХ АЛГОРИТМІВ І ПРОГРАМ

Розділ: Етапи розробки паралельних програм. Паралельні алгоритми

ЛЕКЦІЯ 8. ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ

ПИТАННЯ ДЛЯ ВИВЧЕННЯ

1. Типи паралельних програм
2. Етапи розробки паралельних програм.
3. Планування паралельних програм
4. Реалізація програм
5. Приклад розробки програм.

ТИПИ ПАРАЛЕЛЬНИХ ПРОГРАМ

1. **Ітеративні програми**. Кожна гілка паралельної програми виконує аналогічні дії, наприклад ітерації циклу (множення матриць).
2. **Рекурсивні**. Кожна гілка викликає рекурсивну функцію, при цьому реалізується принцип розділяй і володарюй. Приклад Швидке сортування.
3. **Виробники та споживачі.** (конвеєр). Основна характеристика – вихід попереднього кроку – вхід наступного. Приклад. Обробка даних великого файлу з записом в файл
4. **Клієнт і сервер.** Клієнт посилає запит, а сервер виконує обробку запитів паралельно. Якщо на одному процесорі, то клієнти визивають потрібні функції паралельно. Якщо на різних – видалений виклик функцій або рандеву. Основна характеристика – обмін даними між клієнтом і сервером. Приклад – файлова система та запити програми на введення – виведення.
5. **Керуючи та робочі.** Є один потік, який розподіляє задачі між іншими потоками. Приклад – розподілена система для множення матриць. Потоку передається одна матриця і декілька колонок іншої матриці
6. **Рівні.** Є так званий портфель задач, якій вказує номера рядка і колонки для чергової задачі. Потік обирає наступні дані, збільшує їх номер і виконує наступну задачу.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ

do{

Етап 1. Отримання найбільш ефективного варіанта в послідовному режимі.

Якщо має сенс виконувати паралельно{

Етап 2. Декомпозиція.

Етап 3. Планування паралельних програм.

Етап 4. Реалізація програми і аналіз її продуктивності.

}

} Поки не отримані необхідні показники

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 1

Отримання найбільш ефективного варіанта в послідовному режимі

- Створення послідовного варіанта програми.
- Створення набору тестів.
- Перевірка роботи програми в умовах навантаження, яке досягає максимально можливого для даної задачі.
- Перевірка відсутності витоку пам'яті.
- Перевірка швидкісних характеристик.

ЧИ ТРЕБА ВИКОНУВАТИ ПАРАЛЕЛЬНО?

1. Визначаємо час виконання програми при максимальних значеннях обсягу даних. Якщо час виконання менше ніж 1 с – не виконуємо програму паралельно (чому?).
2. Визначаємо час виконання окремих модулів програми (Профіліровщики, вбудовані засоби, вимір часу)
3. Якщо модуль потребує менше ніж 10% часу на виконання всієї функції – його паралельно виконувати не треба (чому?)
4. Якщо є модулі, які виконувати паралельно – виконуємо наступний етап

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 1

Приклад.

Хай необхідно Скласти функцію для обчислення A = B*C+D*E, де A, B, C, D, E – матриці розміром [n][n].

void MSumMatr (float *a, float *b, float*c, float *d, float *e);// a = b*c + d *e

Визначимо функції для обчислення суми та добутку для матриць.

1. В якості тестів будемо задавати матриці розміром 0, 1, n-1, n, n+1.
2. В функціях не будемо використовувати динамічну пам’ять, тому витоку пам'яті досліджувати не треба.
3. Для визначення швидкісних характеристик будемо використовувати профайлер. Він же використовується для знаходження вузьких міст програми.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. Етап 1

Функція множення матриць.

Розглянути методи:

- множення рядка на колонку;
- рядок на рядок.
- Спеціальні методи

Умови:

Усі матриці одночасно вміщуються в Кеш →краще перший метод.

Усі матриці одночасно не вміщуються в Кеш →краще другий метод.

Підвищені вимоги до швидкодіїї – спеціальні методи.

Хай розмір кешу 2^{15}. Такий кеш вміщує 2^{13} елементів довжиною 4 байта (8192) Для кожної з 5 матриць 1638 елементів. Тобто максимальний розмір квадратних матриць 40 X 40 елементів.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 2

Декомпозиция

Задача. Знайти, що можна виконувати паралельно.

При цьому розглядаються:

Паралелізм по даним. Потік даних розподіляється між окремими задачами.

Паралелізм по функціям. Для усіх функцій, які треба виконати, визначаються ті, які можна виконати паралельно.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 2

Декомпозиція по даним (Domain)

Кількість потоків не обмежена. (Максимальний паралелізм). Одне дане – один потік (теоретичний сенс).

Кількість потоків обмежена.

Одне дане – одна задача. Після завершення обробки потік бере наступну задачу. Найкраще масштабування. Використовує PPL по замовченню.

Всі дані діляться на порції по кількості потоків. Один потік – одна порція даних. Максимальна грануляція. Використовується OPEN MP по замовченню.

Комбінований. Один потік – порція фіксованої довжини. Після завершення обробки – наступну порцію. Використовується, якщо ітерації потребують різного часу. Може використовуватися в Open MP

Комбінований. Один потік – як для другого режиму. Якщо потік закінчив роботу – краде у інших потоків ітерації (PPL, TBB)

Порція для багатовимірних масивів – багато варіантів!!!

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 2

Декомпозиція по задачам (Task)

Мета: визначити функції, які можна виконувати паралельно.

Максимальний паралелізм – одна функція – один потік (кількість функцій обмежена – масштабування, час виконання різних функцій – різний – рівномірне завантаження ядер - балансування).

Один потік – порція функцій, розподіл на порції відповідно часу виконання функцій.

Проблеми: масштабування, балансування.

Приклад. Нехай, є 2 функції. Перша читає дані з диска, а інша - їх обробляє. З – записує результат на диск. Такі функції не можуть виконуватися паралельно.

Що робити?

можна в кожній з функцій читати порцію даних і обробляти порцію після її читання (конвеєр), у цьому випадку функції можна буде виконувати паралельно. Можливість використання конвеєрів є в різних середовищах для створення паралельних програм (TBB, PPL)

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Планування паралельних програм :

Мета: Визначити очікуваний виграш від реалізації для сучасних і перспективних процесорів (I9 – 10 ядер)

- досліджується наявність залежностей для даних і функцій;
- у випадку наявності залежностей визначаються необхідні об'єкти синхронізації. Може бути зроблений висновок про недоцільність паралельного виконання;
- у випадку паралельного виконання час визначається часом виконання самої довгої гілки коду, тому розглядаються паралельні гілки з погляду балансування;
- створення й знищення потоків пов'язане з додатковими витратами, тому необхідно щоб час обробки паралельного потоку перевершував ці витрати (гранулярність);
- час виконання операцій введення – виведення, як правило, не передбачуваний, тому необхідно проаналізувати окремо всі такі операції;
- паралельні алгоритми часто приводять до збільшення необхідних ресурсів, особливо пам'яті. Необхідно проаналізувати реальність вимог. Приклад – паралельне множення матриці – рядок на другу матрицю – друга матриця в кеш пам'яті кожного потоку.
- треба проаналізувати поведінку програми при збільшенні кількості ядер процесору (масштабування) – в зв’язку з тим, що кількість ядер буде весь час збільшуватись;
- треба визначити очікувані значення показників паралельної програми, а саме, прискорення, ефективність, вартість

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Залежності по даним і по керуванню.

Залежність по керуванню – не можна виконувати функцію до тих пір, поки не підготовлені усі аргументи для її обчислення.

Залежності даних. Умови Берстайна.

Набори вхідних і вихідних змінних програми.

Набір вхідних змінних програми R(P) (R від слова read) - це набір вхідних змінних для всіх її операторів.

Набір вихідних змінних програми W(P) (W від слова write) - набір вихідних змінних для всіх її операторів.

Фрагмент програми.

x=u+v; y=x*w;

R(P) = {u, v, x, w},

W(P) = {x, y}.

Помітимо, що змінна x присутня як в R(P), так і в W(P).

Умови Бернстайна.

- якщо для двох ділянок програм P і Q:
- перетинання W(P) і W(Q) порожньо,
- перетинання W(P) з R(Q) порожньо,
- перетинання R(P) і W(Q) порожньо,

Висновок: залежностей гарантовано немає, ділянки програм можна виконувати паралельно.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Приклад (умови Бернстайна)

Перевіримо умови Бернстайна для двох фрагментів програми P, Q:

P:	***Q:***
x=2	*x=3*
y=x-1	*y=x+1*

1. Визначимо множини:

R(P) = {x}	*R(Q) = {x}*
W(P) = {x, y}	*W(Q) = {x, y}*

2. Аналізуємо залежності:

$W(P) \cap W(Q) = \{x, y\}$

Висновок.

Результат може бути не детермінованим при виконанні ділянок програми *P* і *Q* паралельно. Якщо ділянки *P* і *Q* виконувати в критичній секції, то паралельне виконання припустиме.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Типи залежностей.

- Залежність типу **«Читання після запису», RAW** (i: W, j: R, i<j)
- Залежність типу «**Запис після запису**», **WAW** (i: W, j: W, i<j)
- Залежність типу «**Запис після читання** », **WAR** (i: R, j: W, i<j)
- **WAW та WAR** називаються анти залежностями, їх легко уникнути, якщо для попереднього й наступного значень змінної використовувати різні змінні.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Типи залежностей.

Нехай необхідно обчислити значення площі трикутника: $S = \sqrt{p(p-a)(p-b)(p-c)}$

Очевидно, що паралельно можна виконувати 3 операції

Використаємо оператори для обчислення:

1	*s=p-a;*	*залежності немає тільки між*	
2	***s = s*p;***	*операторами 2 і 3*	*RAW, WAR*
3	*r1 = p – b;*		
4	*s = **s** *r1;*		*RAW, RAW, WAR*
5	*r1 = p-c;*		*WAR*
6	*s = s * r1;*		*RAW,RAW, WAR*
7	*s =sqrt (s);*		*RAW, WAR*

Потребує 6 операцій

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Аналіз залежностей.

Залежність між *r1 = p – a;* і *r1 = p * r1;* оператором типу «Читання після запису» - усунути не можна.

Залежність між *r1 = p – a;* і *r1 = p * r1;* оператором типу «Запис після читання» - усунути можна, якщо використовувати інше ім’я для результату.

Залежність між *r2 = p – b;* і *r2 = r2 * r3;* оператором типу “ Читання після запису» - усунути не можна.

Залежність між *r2 = p – b;* і *r2 = r2 * r3;* оператором типу “Запис після читання» - усунути можна, якщо використовувати інше ім’я для результату.

Залежність між *r3 = p – c;* і *r2 = r2 * r3;* оператором типу “Запис після читання» - усунути можна, якщо використовувати інше ім’я для результату.

Залежність між *r1 = p * r1;* і *r1 = r1 * r2;* оператором типу “Читання після читання» - усунути не можна.

Залежність між *r1 = r1 * r2;* і *S =sqrt (r1);* оператором типу “Читання після читання» - усунути не можна.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Аналіз залежностей.

Перетворення програми для забезпечення мінімальної кількості залежностей

1 r1 = p – a;

2 r2 = p – b;

3 r3 = p – c;

*4 r4 = p * r1;* Залежить від 1

*5 r5 = r2 * r3;* Залежить від 2 та 3

*6 r6 = r4 * r5;* Залежить від 4 та 5

7 S =sqrt (r6); Залежить від 6

Потребує 4 операції

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Масиви та залежності.

Ділянка програми:

x [a * i1 + b] = …; //W

…

F = x [c * i2 + d]; //R

Умова наявності залежності.

a * i1 + b == c * i2 + d → a * i1 + (-c)*i2 = d – b;

*a' * i1 + c'*i2 = d',*

де

a' = a; c' = -c; d'= d - b.

a'*i1 + c'*i2 = *d'* - **лінійне діафантове рівняння**

Умова наявності коренів цілого типу (індекси завжди цілі!!!):

d' %GCD(a', c') ==0, тобто *(d – b) % GCD (a, c) ==0*

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Рішення діофантового рівняння:

Для знаходження всіх коренів рівняння $a*x + b*y = c$ використовуються формули:

$$r = GCD(a, b);$$

$$x = x_0 - \frac{b}{r} n;$$

$$y = y_0 + \frac{a}{r} n;$$

- Тут: { x_0,y_0} – старе рішення рівняння;
- n – будь-яке ціле.
- {x, y} – нове рішення.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Приклади GCD тестування. Приклад 1

Нехай значення індексів:

*2 * i + 1;*

*4 * j + 2*.

У цьому випадку залежностей бути не може, тому що значення першого індексу непарне, а значення другого - парне.

Перевіримо це за допомогою *GCD* тесту. *GCD* (2, 4) = 2; 2 – 1 = 1; 1 не ділиться на 2.

Залежностей немає.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Приклади GCD тестування. Приклад 2

$$r=GCD(a,b);$$

$$x=x_0-\frac{b}{r}n;$$

$$y=y_0+\frac{a}{r}n;$$

for (i = 0; i < 100; ++i) x [3 i + 5] = …;*

*for (j = 0; j < 100; ++j) {r = x[5 * j + 3];…}*

Хай кожний з двох потоків виконує свій цикл.

3 i + 5 = 5 * j + 3 при i = j = 1.*

Залежність є.

Дійсно, значення *r при j = 1* залежить від того виконається чи ні перший цикл для i=1.

Перевірка за тестом *GCD* .

GCD (3, 5) = 1; 5 – 3 = 2; 2 ділиться на 1. Таким чином, залежність може бути.

Визначимо, при яких значеннях індексів *i, j* значення виразів *3* i + 5 і 5 * j + 3* збігаються.

Запишемо умову у формі рівняння Діофанта : *3i – 5j = -2*.

Очевидно, що корінь рівняння є: *i = 1, j = 1*.

Інші корені рівняння Діофанта: {6, 4}; {11, 7};…

Висновок: *залежності є*

Для багатовимірних масивів – перевірка по значенню приведеного індексу

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Видалення залежностей. Приклади.

Приклад 1.

```
for (int i = 0; i < 100; i++)
{
    a [i]+= b [i];                  // a
    b [i + 1] = c [i] + d [i];      // b
}
```

Якщо i+1 ітерація виконається перед i-ой, то отримаємо RAW, тобто залежність не можна видалити зміною імені.

Перевірка наявності залежності по GCD тесту.

i+1, j gcd (1, 1) = 1; d-b = 1; залежність є.

Перетворення коду:

```
a [0]+= b [0];
#pragma omp parallwl for
for (int i = 1; i < 100; i++){
        b [i] = c [i-1] + d [i-1];          a [i]+= b [i];
}
b [100] = c [99] + d [99];
```

Залежності немає!!!

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Видалення залежностей. Приклад 2.

```
for (int i = 1; i < N; i++)
{
      for (int j = 1; j < N; j++)
            a [i][j] = a [i-1][j - 1] * a [i-1][j - 1];
}
```

Якщо i+1 ітерація виконається перед i-ой, то отримаємо RAW, тобто залежність не можна видалити зміною імені.

Перетворення коду:

```
for (int i = 1; i < N; i++)
{
     #pragma omp parallel for
     for (int j = 1; j < N; j++)
            a [i][j] = a [i-1][j - 1] * a [i-1][j - 1];
}
```

Залежності немає!!!

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Планування паралельних програм :

- досліджується наявність залежностей для даних і функцій;
- **у випадку наявності залежностей визначаються необхідні об'єкти синхронізації. Може бути зроблений висновок про недоцільність паралельного виконання;**
- у випадку паралельного виконання час визначається часом виконання самої довгої гілки коду, тому розглядаються паралельні гілки з погляду балансування;
- створення й знищення потоків пов'язане з додатковими витратами, тому необхідно щоб час обробки паралельного потоку перевершував ці витрати (гранулярність);
- час виконання операцій введення – виведення, як правило, не передбачуваний, тому необхідно проаналізувати окремо всі такі операції;
- паралельні алгоритми часто приводять до збільшення необхідних ресурсів, особливо пам'яті. Необхідно проаналізувати реальність вимог. Приклад – паралельне множення матриці – рядок на другу матрицю – друга матриця в кеш пам'яті кожного потоку.
- треба проаналізувати поведінку програми при збільшенні кількості ядер процесору (масштабування) – в зв’язку з тим, що кількість ядер буде весь час збільшуватись;
- треба визначити очікувані значення показників паралельної програми, а саме, прискорення, ефективність, вартість

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Синхронізація.

1 Бар'єр (Barrier). Використовується для всіх паралельних задач, якщо необхідно забезпечити продовження коду в даному місці тільки після завершення всіх паралельних ділянок коду, наприклад, усіх ітерацій циклу, які виконуються паралельно.

X = f1(…) ; Y = f2 (…)

Barrier

Z = …X…Y

WaitFor…

2 ***Мьютекс (mutex).*** Використовується для захисту загальних даних (критичні секції). Перша задача встановлює «замок». Інші задачі чекають, поки власник «замка» не відкриє його.

3 Якщо були потрібні операції комунікації (використовується розподілена пам'ять), то може знадобитися ***синхронізація для операцій комунікації***. Необхідно забезпечити, щоб до завершення передачі даних, вони не почали оброблятися. Після завершення обробки порції даних, результат повинен бути переданий приймачу до того, як почнеться обробка нової порції даних.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Планування паралельних програм :

- досліджується наявність залежностей для даних і функцій;
- у випадку наявності залежностей визначаються необхідні об'єкти синхронізації. Може бути зроблений висновок про недоцільність паралельного виконання;
- **у випадку паралельного виконання час визначається часом виконання самої довгої гілки коду, тому розглядаються паралельні гілки з погляду балансування;**
- створення й знищення потоків пов'язане з додатковими витратами, тому необхідно щоб час обробки паралельного потоку перевершував ці витрати (гранулярність);
- час виконання операцій введення – виведення, як правило, не передбачуваний, тому необхідно проаналізувати окремо всі такі операції;
- паралельні алгоритми часто приводять до збільшення необхідних ресурсів, особливо пам'яті. Необхідно проаналізувати реальність вимог. Приклад – паралельне множення матриці – рядок на другу матрицю – друга матриця в кеш пам'яті кожного потоку.
- треба проаналізувати поведінку програми при збільшенні кількості ядер процесору (масштабування) – в зв’язку з тим, що кількість ядер буде весь час збільшуватись;
- треба визначити очікувані значення показників паралельної програми, а саме, прискорення, ефективність, вартість

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Балансування.

Хай необхідно виконати цикл (n ітерацій).

Хай кількість паралельних гілок m

Хай час виконання ітерацій однаковий для усіх ітерацій.

1. Один потік – ⌜ n/m ⌝ ітераций (16/ 5 потоків, 4, 4, 4, 4 – один процесор не зайнятий під час виконання 4 ітерацій, загальний час виконання як для 4 ітерацій).
2. Один потік –n/m ітераций (16 / 5 потоків, 3, 3, 3, 3, 3, 1 – чотири процесора не зайняті під час виконання однієї ітерації, загальний час виконання як для 4 ітерацій)
3. 1 потік: 4 ітерації
   2 потік: (16 - 4) /4 =3
   3 потік: (12 – 3)/3 = 3
   4 потік: (9 – 3)/2 = 3
   5 потік: (6 - 3)/ 1 = 3

Хай час виконання ітерацій не однаковий для усіх ітерацій.

16 ітерацій – 16 задач, кожному потоку 3 ітерації, якщо потік завершив роботу – йому додаткову ітерацію, якщо закінчив і більше немає додаткових, краде у інших потоків

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Планування паралельних програм :

- досліджується наявність залежностей для даних і функцій;
- у випадку наявності залежностей визначаються необхідні об'єкти синхронізації. Може бути зроблений висновок про недоцільність паралельного виконання;
- у випадку паралельного виконання час визначається часом виконання самої довгої гілки коду, тому розглядаються паралельні гілки з погляду балансування;
- **створення й знищення потоків пов'язане з додатковими витратами, тому необхідно щоб час обробки паралельного потоку перевершував ці витрати (гранулярність);**
- час виконання операцій введення – виведення, як правило, не передбачуваний, тому необхідно проаналізувати окремо всі такі операції;
- паралельні алгоритми часто приводять до збільшення необхідних ресурсів, особливо пам'яті. Необхідно проаналізувати реальність вимог. Приклад – паралельне множення матриці – рядок на другу матрицю – друга матриця в кеш пам'яті кожного потоку.
- треба проаналізувати поведінку програми при збільшенні кількості ядер процесору (масштабування) – в зв’язку з тим, що кількість ядер буде весь час збільшуватись;
- треба визначити очікувані значення показників паралельної програми, а саме, прискорення, ефективність, вартість

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Гранулярність.

- Гранулярність – це визначення мінімального блоку, для якого має сенс використовувати потік.
- Для збільшення продуктивності бажано виконувати блок максимального розміру.
- Експериментальний алгоритм визначення гранулярності для отримання заданого прискорення S_0 (n – загальна кількість ітерацій):

```
block = 1;
while (1)
{
    // Виконання операції;
    // Визначення прискорення S
    if (S> S0) break;
     block*=2;
}
```

Якщо block>=n, то треба змінити алгоритм обчислення

Збільшення кількості ядер

#pragma omp parallel for

for () Перешли з i3 (2 ядра) на i9 (10 ядер) – розмір порції в 5 разів менше – зменшити кількість ядер, які використовувати!!!

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Планування паралельних програм :

- досліджується наявність залежностей для даних і функцій;
- у випадку наявності залежностей визначаються необхідні об'єкти синхронізації. Може бути зроблений висновок про недоцільність паралельного виконання;
- у випадку паралельного виконання час визначається часом виконання самої довгої гілки коду, тому розглядаються паралельні гілки з погляду балансування;
- створення й знищення потоків пов'язане з додатковими витратами, тому необхідно щоб час обробки паралельного потоку перевершував ці витрати (гранулярність);
- **час виконання операцій введення – виведення, як правило, не передбачуваний, тому необхідно проаналізувати окремо всі такі операції;**
- паралельні алгоритми часто приводять до збільшення необхідних ресурсів, особливо пам'яті. Необхідно проаналізувати реальність вимог. Приклад – паралельне множення матриці – рядок на другу матрицю – друга матриця в кеш пам'яті кожного потоку.
- треба проаналізувати поведінку програми при збільшенні кількості ядер процесору (масштабування) – в зв’язку з тим, що кількість ядер буде весь час збільшуватись;
- треба визначити очікувані значення показників паралельної програми, а саме, прискорення, ефективність, вартість

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Врахування операцій введення – виведення.

Приклад. Анті плагіат. Зміст одного файлу порівнюється з усіма наступними.

Операції введення - виведення можуть істотно сповільнити виконання ітерації.

Асинхронне введення - виведення залежить від платформи, тому використовувати його для програм, які повинні працювати на різних платформах, не рекомендується.

Необхідно пам'ятати, що **операції запису в той самий файл** різними паралельними гілками повинні виконуватися в ексклюзивному режимі, операції читання можуть виконуватися одночасно з іншими операціями читання.

Якщо введення - виведення даних виконується з **мережевих ЕОМ**, то не можна прогнозувати час, необхідний для доступу до таких даних.

Є **паралельні файлові системи**, які, на рівні файлової системи, підтримують паралельний файловий доступ. Приклади таких систем: PVFS/PVFS2, Linux; GPFS: General Parallel File System для AIX (IBM). Але, на жаль, жодна з файлових систем, які використовуються з Windows, не є паралельною.

Рекомендації.

Рекомендується операції введення - виведення виконувати в послідовному режимі. Якщо це не виходить, то кожна гілка хай використовує свій файл, а потім результати роботи можна об'єднати.

Якщо потік виконує операції введення – виведення, рекомендується, щоб кількість потоків перевищувала кількість ядер, для компенсування блокувань при введенні – виведенні.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Планування паралельних програм :

- досліджується наявність залежностей для даних і функцій;
- у випадку наявності залежностей визначаються необхідні об'єкти синхронізації. Може бути зроблений висновок про недоцільність паралельного виконання;
- у випадку паралельного виконання час визначається часом виконання самої довгої гілки коду, тому розглядаються паралельні гілки з погляду балансування;
- створення й знищення потоків пов'язане з додатковими витратами, тому необхідно щоб час обробки паралельного потоку перевершував ці витрати (гранулярність);
- час виконання операцій введення – виведення, як правило, не передбачуваний, тому необхідно проаналізувати окремо всі такі операції;
- **паралельні алгоритми часто приводять до збільшення необхідних ресурсів, особливо пам'яті. Необхідно проаналізувати реальність вимог.**
- треба проаналізувати поведінку програми при збільшенні кількості ядер процесору (масштабування) – в зв’язку з тим, що кількість ядер буде весь час збільшуватись;
- треба визначити очікувані значення показників паралельної програми, а саме, прискорення, ефективність, вартість

АНАЛІЗ ПОТРІБНИХ РЕСУРСІВ

Хай треба паралельно обчислити добуток 2 поліномів:

$(a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0) * (b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \ldots + b_1x + a_0) =$

$(c_{2n-1}x^{2n-1} + c_{2n-2}x^{2n-2} + \ldots + c_1x + c_0)$

При паралельному обчисленні для кожної порції обчислюємо частковий добуток, який складається з 2n елементів, які треба зберігати в пам'яті окремо для кожного потоку. В кінці їх треба додати.

Таким чином додаткова пам’ять, необхідна для паралельного режиму (m – 1)* 2n елементів. Якщо для їх зберігання треба використовувати диск – паралельні обчислення не ефективні!

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Планування паралельних програм :

- досліджується наявність залежностей для даних і функцій;
- у випадку наявності залежностей визначаються необхідні об'єкти синхронізації. Може бути зроблений висновок про недоцільність паралельного виконання;
- у випадку паралельного виконання час визначається часом виконання самої довгої гілки коду, тому розглядаються паралельні гілки з погляду балансування;
- створення й знищення потоків пов'язане з додатковими витратами, тому необхідно щоб час обробки паралельного потоку перевершував ці витрати (гранулярність);
- час виконання операцій введення – виведення, як правило, не передбачуваний, тому необхідно проаналізувати окремо всі такі операції;
- паралельні алгоритми часто приводять до збільшення необхідних ресурсів, особливо пам'яті. Необхідно проаналізувати реальність вимог.
- **треба проаналізувати поведінку програми при збільшенні кількості ядер процесору (масштабування) – в зв’язку з тим, що кількість ядер буде весь час збільшуватись;**
- треба визначити очікувані значення показників паралельної програми, а саме, прискорення, ефективність, вартість

ВРАХУВАННЯ ХАРАКТЕРИСТИК КОМП'ЮТЕРУ

Збільшення кількості ядер

#pragma omp parallel for

for () Перешли з i3 (2 ядра) на i9 (10 ядер) – розмір порції в 5 разів менше – зменшити кількість ядер, які використовувати!!!

Попереднє на етапі грануляції було визначено мінімальну кількість потоків, при якій паралельний режим має ефект.

При збільшенні кількості ядер при автоматичному розподілу ітерацій кількість ітерацій на один потік може бути не ефективним.

Якщо кількість ітерацій має критичне значення треба обмежити кількість потоків, які використовуються.

Кількість потоків = min (Кількість ядер, max (Кількість потоків)).

Приклад. Хай для попереднього прикладу паралельний режим має сенс для ступеня поліному 128 і більше.

Визначимо максимальну кількість потоків, яка має сенс:

max (Кількість потоків) = n/128;

Хай кількість ядер дорівнює 4.

Визначити кількість потоків, які має сенс використовувати для n = 256:

Кількість потоків = min (Кількість ядер, max (Кількість потоків)) = min (4, 256 / 128) = 2, таким чином кількість потоків треба обмежити 2.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 3

Планування паралельних програм :

- досліджується наявність залежностей для даних і функцій;
- у випадку наявності залежностей визначаються необхідні об'єкти синхронізації. Може бути зроблений висновок про недоцільність паралельного виконання;
- у випадку паралельного виконання час визначається часом виконання самої довгої гілки коду, тому розглядаються паралельні гілки з погляду балансування;
- створення й знищення потоків пов'язане з додатковими витратами, тому необхідно щоб час обробки паралельного потоку перевершував ці витрати (гранулярність);
- час виконання операцій введення – виведення, як правило, не передбачуваний, тому необхідно проаналізувати окремо всі такі операції;
- паралельні алгоритми часто приводять до збільшення необхідних ресурсів, особливо пам'яті. Необхідно проаналізувати реальність вимог.
- треба проаналізувати поведінку програми при збільшенні кількості ядер процесору (масштабування) – в зв’язку з тим, що кількість ядер буде весь час збільшуватись;
- **треба визначити очікувані значення показників паралельної програми, а саме, прискорення, ефективність, вартість**

ОБЧИСЛЕННЯ ПОКАЗНИКІВ

1. Обрати закон для обчислення параметрів: Амдаля або Густафсона.
2. Визначити частку програми, яку треба виконувати послідовно (експериментально).
3. Обчислити критерії: прискорення, ефективність, та вартість.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ЕТАП 4

Реалізація програми й аналіз її продуктивності.

Реалізація паралельної програми.

Перевірка правильності її роботи для усіх тестів, розроблених для послідовного варіанту.

Перевірка наявності race condition та deadlock (Parallel Studio).

Експериментальне визначення прискорення, ефективності та вартості. Порівняння з очікуваними значеннями.

Якщо параметри, які вимагаються, не досягнуті, то повторно виконується блок планування, а, можливо, і декомпозиції.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ПРИКЛАД. ЕТАП 1

Приклад.

Нехай необхідно обчислити елементи двомірного масиву. Для обчислення використовувати функцію, що залежить тільки від номерів індексів цього масиву.

У послідовному режимі програма має вигляд:

```
void CreateMassiv (float ar [][MAXSIZE], int n)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            ar [i][j] = sqrt (double (i * i + j)) +
            sin (double(i * i - j));
}
```

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ПРИКЛАД. ЕТАП 2

Крок 1. Декомпозиція. Для множини даних використовується одна й та ж функція, тому будемо виконувати **декомпозицію за даними**. Розіб'ємо результуючий масив на рядки, тому що припускаємо розміщення елементів масиву по рядках. У кожну порцію виділимо $\lceil n/P \rceil$ рядків (P – кількість процесорів. Кожний процесор працює зі своїм шаром рядків.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ПРИКЛАД. ЕТАП 3

Залежності. Залежностей за даними в задачі немає. Паралельне виконання можливо.

Синхронізація. Загальні дані не використовуються. Синхронізація не потрібна. Перед завершенням програми чекати завершення всіх ітерацій.

Балансування. Час виконання однієї ітерації не залежить від її номера. Кількість ітерацій може бути не кратною *P*. Розподілимо ітерації таким чином: спочатку кожний потік буде виконувати n/P ітерацій, потім. залишок ітерацій - по одному на один процесор, в цьому разі час виконання обчислень визначається часом виконання порції розміром ⌐n/P¬ ітерацій одним процесором.

Гранулярність. Експериментально визначаємо максимальну кількість рядків, для якої паралельний режим виконання не має переваг. Якщо після обчислення розмір порції даних виявляється менше визначеної величини, то укрупнюємо порції (повернення до кроку 1)

Облік операцій введення - виведення. Операцій введення - виведення немає.

Визначення необхідних ресурсів. Для цієї задачі не потрібно додаткових ресурсів для паралельного режиму. Виключення - якщо використовуються системи з розподіленою пам'яттю, тоді результати бажано одержувати в пам'яті «свого» процесора, а наприкінці переслати в загальну пам'ять.

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ПРИКЛАД. ЕТАП 3

Визначення очікуваних показників програми з урахуванням паралельного виконання

Визначимо максимальне прискорення.

Для даної задачі послідовно нічого виконувати не треба, тому β = 0, максимальне прискорення не обмежене.

Хай час виконання однієї ітерації дорівнює t. У цьому випадку

$T(1) = t * n * n$;

$T(p) = t^* (n * (n /p + 1))$.

$S = T(1)/ T(p) = (t * n * n)/(t^* (n *(n /p + 1))) = (n * p)/ (n + p)$;

$\lim_{n\to\infty} S = p$;

$E = S / p = n/ (n + p)$;

$\lim_{n\to\infty} E = 1$;

$C= T(p) * p = t^* n * (n + p)$

ЕТАПИ РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ. ПРИКЛАД. ЕТАП 4

Паралельна функція (Open MP)

```
void ParallelCreateMassiv (float ar [][MAXSIZE],
    int n)
{
#pragma omp parallel for
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            ar [i][j] = sqrt (double (i * i + j)) +
                sin (double(i * i - j));}
}
```

ВИСНОВКИ

- Перед розробкою паралельної версії програми необхідно створити відповідну послідовну програму. Це дозволяє визначити, чи потрібно розробляти паралельну версію, чи немає протиріч в вимогах, визначити необхідний набір тестів та початкові значення для визначення параметрів ефективності паралельних програм.
- На етапі проектування, перш за все, аналізуються залежності для визначення, що можна виконувати паралельно. Далі визначається кількість паралельних гілок та розподіл завантаження між цими гілками для забезпечення масштабування та гранулярності. Визначаються теоретичні значення прискорення, ефективності та вартості. Цей етап виконується до тих пір, поки не досягнуті показники, які відповідають вимогам до програм.
- Останній етап - реалізація паралельного алгоритму. Після реалізації необхідно не тільки визначити критерії ефективності, але і проаналізувати програму на наявність проблем, пов'язаних з race conditions, deadlocks.
- Якщо після виконання останнього етапу не досягнуті необхідні результати, обирається інша технологія формування паралельної програми (Open MP, PPL, TBB, …), або етапи 1-4 повторюються.

ПИТАННЯ ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ

1. Технології розробки послідовних програм http://ru.wikipedia.org/wiki/Разработка_программного_обеспечения.
2. Граф програми. Еквівалентні перетворення програми (Воєводин)

МАТЕРІАЛИ ДЛЯ ЕКСПРЕС-КОНТРОЛЮ

1. Чи завжди послідовне та паралельне виконання необхідно виконувати за одним і тим же алгоритмом?
2. Яка мета створення послідовної програми, якщо алгоритми відрізняються?
3. Чи гарантує наявність залежностей в коді, якщо умови Берстайна дають позитивний результат?
4. Чи гарантує наявність залежностей в коді, якщо GCD тест дає позитивний результат?
5. Яким чином можна використовувати GCD тест для багато вимірних масивів?
6. Перевірити наявність залежностей для коду *for (size_t i = 0; i < n; i++) x [i] = y [i];*відповідно GCD тесту
7. Від чого залежить Гранулярність?
8. Яким чином досягається властивість Масштабування?
9. Від чого залежить кількість потоків, яка обирається для вирішення задачі?