СУЧАСНІ ПРОЦЕСОРИ ТА ПАРАЛЕЛІЗАЦІЯ ОБЧИСЛЕНЬ

Лекція 2. Передбачення переходів

ТИПИ ПАРАЛЕЛІЗМУ

1. Паралелізм на рівні бітів.
2. **Паралелізм на рівні команд. Конвеєр.**
3. **Паралелізм на рівні команд. Спекулятивне виконання.**
4. **Паралелізм на рівні команд. Суперскалярність.**
5. **Паралелізм на рівні даних (цикли обробки даних).**
6. **Паралелізм на рівні функцій (функції. Які виконуються паралельно).**
7. **Паралелізм на рівні програм**

ТИПИ ПАРАЛЕЛІЗМУ. КОНВЕЄР

$T1 = n*t$

$T2 = t + (n - 1)*t/m$

n – кількість команд для виконання;

t – час виконання команди без конвеєру;

T1 – час виконання програми без конвеєру

T2 – час виконання програми з конвеєром

$n \uparrow$ $\qquad\qquad T1 \approx T2 * m$

ТИПИ ПАРАЛЕЛІЗМУ. КОНВЕЄР

Конвеєр та команди переходів. Хай конвеєр має 5 блоків. **c = max (a, b)**

```
mov eax, [a]
cmp eax, [b]
jge short m1
mov eax, [b]
M1:
mov [c]. eax
```

Стадія 1	Стадія 2	Стадія 3	Стадія 4	Стадія 5
mov eax, [a]				
	Mov			
		eax, [a]		
			Eax , [a]	
				Eax = [a]

ТИПИ ПАРАЛЕЛІЗМУ. КОНВЕЄР

Конвеєр та команди переходів. Хай конвеєр має 5 блоків. **c = max (a, b)**
mov eax, [a]
cmp eax, [b]
jge short m1
mov eax, [b]
M1:
mov [c]. eax

Стадія 1	**Стадія 2**	**Стадія 3**	**Стадія 4**	**Стадія 5**
mov eax, [a]				
cmp eax, [b]	mov			
	cmp	eax, [a]		
		eax, [b]	Eax , [a]	
			Eax><== [b]	Eax = [a]
				Eflags
Розмір конвеєру:більше 20 для останніх одноядерних (Pentium 4), 14 для i7				

ТИПИ ПАРАЛЕЛІЗМУ. КОНВЕЄР

Конвеєр та команди переходів. Хай конвеєр має 5 блоків. **c = max (a, b)**

```
mov    eax, [a]
cmp eax, [b]
jge short m1
mov    eax, [b]
M1:
mov    [c]. Eax
```

Стадія 1	Стадія 2	Стадія 3	Стадія 4	Стадія 5
1 mov eax, [a]				
2 cmp eax, [b]	mov			
3 jge short m1	Cmp	eax, [a]		
4 Mov eax, [b]	jge	eax, [b]	Eax , [a]	
5	mov	m1	Eax><== [b]	Eax = [a]
6		Eax, [b]		Eflags
7 jge short m1				
8 Mov [c], eax	jge			
9	mov	m1		
10		[c], eax	M1	
11			[c]. Eax	
				c

ЧОМУ ВАЖЛИВО РОЗУМІТИ КОНВЕЄР?

Приклад. В заданому масиві обчислити кількість парних та непарних чисел

Скласти функцію та визначити обчислювальну складність для наступних варіантів:

1 Усі числа масиву непарні;

2 усі числа масиву парні;

3 Числа випадкові. Імовірність чисел різної парності однакова

ЧОМУ ВАЖЛИВО РОЗУМІТИ КОНВЕЄР?

Функція

```
size_t EvenOddCount1(unsigned *x, size_t &OddCount, size_t n){
    size_t Odd = 0;
    for (size_t i = 0; i < n; i++)
    if (x[i] % 2)
            Odd++;
    OddCount = Odd; return n - Odd;
}
```

EvenCount < OddCount < EvenOddCount

EvenOddCount1: even = 16777216 odd = 0 time = 21.0823

ЧОМУ ВАЖЛИВО РОЗУМІТИ КОНВЕЄР?

Функція

```
size_t EvenOddCount1(unsigned *x, size_t &OddCount, size_t n){
    size_t Odd = 0;
    for (size_t i = 0; i < n; i++)
    if (x[i] % 2)
            Odd++;
    OddCount = Odd; return n - Odd;
}
```

EvenCount < OddCount < EvenOddCount

EvenOddCount1: even = 16777216 odd = 0 time = 21.0823

EvenOddCount1: even = 0 odd = 16777216 time = 11.1155 + конвеєр

ЧОМУ ВАЖЛИВО РОЗУМІТИ КОНВЕЄР?

Функція

```
size_t EvenOddCount1(unsigned *x, size_t &OddCount, size_t n){
    size_t Odd = 0;
    for (size_t i = 0; i < n; i++)
    if (x[i] % 2)
            Odd++;
    OddCount = Odd; return n - Odd;
}
```

EvenCount < OddCount < EvenOddCount

EvenOddCount1: even = 16777216 odd = 0 time = 21.0823

EvenOddCount1: even = 0 odd = 16777216 time = 11.1155

EvenCountOdd1: even = 8388608 odd = 8388608 time = 83.7533 + передбачення

Висновок: Якщо знаємо, імовірність яких чисел більше – їм виконання без переходів!!!

ЧОМУ ВАЖЛИВО РОЗУМІТИ КОНВЕЄР?

2 варіант

```
size_t EvenOddCount2(unsigned *x, size_t &OddCount, size_t n) {
    size_t Odd = 0;
    for (size_t i = 0; i < n; i++)
        Odd+= x[i] % 2;
    OddCount = Odd;
    return n - Odd;
}
```

EvenOddCount2: even = 16777216 odd = 0 time = 7.1774 (21.0823)

EvenOddCount2: even = 0 odd = 16777216 time = 7.04911 (11.1155)

EvenOddCount2: even = 8388608 odd = 8388608 time = 7.04783 (83.7533)

Висновки:

1 Якщо є можливість – видалити команди переходів!

2 Якщо знаємо, імовірність яких даних більше – їм виконання без переходів!!!

Реальна статистика: 80% - звичайні команди, 20% команди переходу!!!

ТИПИ ПАРАЛЕЛІЗМУ. КОНВЕЄР

БПП. Статичне передбачення

- Виконується для команди переходу, яка виконується вперше.
- Результат заноситься в БПП.
- Передбачається відсутність переходу для переходу вперед і наявність для переходу назад

БПП. Динамічне передбачення (з урахуванням історії)

- Виконується для команд переходу, інформація для якої в БПП.
- Інформація коректується.
- Перехід прогнозується в залежності від попередніх результатів

Блок передбачення:

Адреса команди переходу, флаги (адреса – тільки молодші біти)

ТИПИ ПЕРЕХОДІВ

безумовні

- goto (jmp label)
- switch (jmp label[reg])
- Функції звичайні та віртуальні

```
Mov     eax, tabl[reg]
Jmp     eax
```

- Обробники виключень

try{}catch…

Умовні

if(a) block

```
if (!a) goto lab1;
block
lab1:
```

if(a)block1; else block2

```
if(!a)goto lab1;
block1; goto lab2;
lab1:
block2
lab2:
```

ТИПИ ПЕРЕХОДІВ

Цикли

```
for (e1;e2;e3) block;
e1;
goto lab1;
lab2:
e3;
lab1:
If (!e2) goto lab3;
block;
goto lab2;
Lab3:
```

```
Lab2:
while (e1)block;
if (!e1)goto lab1;
block;
Goto lab2;
lab1:

do {block }while (e1);
lab1:
block;
if (e1) goto lab1;
```

АЛГОРИТМИ ПЕРЕДБАЧЕНЬ ДЛЯ УМОВНИХ ПЕРЕХОДІВ

1 біт.
Передбачається відсутність переходу, якщо біт -0, наявність-1.
Біт встановлюється в 1, якщо перехід є.
Біт встановлюється в 0, якщо переходу немає.
Будемо позначати 1 – якщо перехід є, 0 – якщо немає.
E (error) – помилка в передбаченні, o (ok) – правильне передбачення.

```
bool Predic (int F)          int SetFlags (){          int ClearFlag(){
{
        return F& 1                  return 1                  return 0;

}                            }                         }
```

Приклад 1
1 1 1 0 1 1 1 0 1 1 1 .
Якщо немає передбачень – 9 помилок
Є передбачення – 5 помилок
Найгірший варіант: 1 0 1 0,…

АЛГОРИТМИ ПЕРЕДБАЧЕНЬ ДЛЯ УМОВНИХ ПЕРЕХОДІВ

2 біта, рахівник з насищенням (0, 1, 2, 3)

Передбачається відсутність переходу, якщо значення рахівника <2. Інакше наявність.

При наявності переходу значення рахівника збільшується на один, інакше зменшується (з урахуванням насиченості!!!)

```
bool Predic (int F) {          int SetFlags (int F){          int ClearFlag(int F){
        return F>==2;          return (F+= (F != 3));         return (F-= (F != 0);
}                              }                              }
```

Приклад 1

Якщо немає передбачень – 9 помилок

1 1 1 0 1 1 1 0 1 1 1

Рахівник 2 бітний

Є передбачення – 4 помилки

Найгірший варіант: 110101010…

АЛГОРИТМИ ПЕРЕДБАЧЕНЬ ДЛЯ УМОВНИХ ПЕРЕХОДІВ

- **Алгоритм з рахівником з насиченням**

Розглянемо на прикладі рахівника з n бітами:

```
// Прогнозування
BOOL Prediction (BYTE bFlags, int n){
        return bFlags >=(1 << (n – 1));
} // Встановлення бітів
BYTE SetFlags (BYTE bFlags, int n){
        return bFlags += bFlags <((1 << n) – 1) )
                bFlags --;
}

BYTE ClrFlags (BYTE bFlags, int n){
        return bFlags -= bFlags >0;

}

return bFlags;}
```

100001

00		
1	E	01
0	OK	00
0	OK	00
0	OK	00
0	OK	00
1	E	01

2/3 OK

11101110

1	E	01
1	E	10
1	OK	11
0	E	10
1	OK	11
1	OK	11
1	OK	11
0	OK	10

5/8 - OK

АЛГОРИТМИ ПЕРЕДБАЧЕНЬ

- 2 – рівневе адаптивне передбачення

Таблиця станів, кількість елементів визначається n (n = 2 – 4 стани).

Для кожного стану ставиться в відповідність регістр з насиченням. Для визначеного стану (попередні n бітів) прогнозується стан згідно його регістру. Стан регістру корегується, якщо потрібно.

Для реальних процесорів n = 4, 16 рядків таблиця, рахівник k = 2 біта, поточний стан 4 біта, всього 16 * 2 + 4 бітів для кожної команди переходу

(локальний режим).

Глобальний – загальна таблиця

(00) 1 0 0 1 1 0 0 1

N = 2

K = 1

АЛГОРИТМИ ПЕРЕДБАЧЕНЬ

- 2 – рівневе адаптивне передбачення

Таблиця станів, кількість елементів визначається n (n = 2 – 4 стани).

Для кожного стану ставиться в відповідність регістр з насиченням. Для визначеного стану (попередні n бітів) прогнозується стан згідно його регістру. Стан регістру корегується, якщо потрібно.

Для реальних процесорів n = 4, 16 рядків таблиця, разхівник 2 біта, поточний стан 4 біта, всього 16 * 2 + 4 бітів для кожної команди переходу

(локальний режим).

Глобальний – загальна таблиця

(00) 1 0 0 1 1 0 0 1
E O O O O O O O

Регістр зсуву

00 01 10 0 0 01 11 10 0 0 01

Таблиця:

00	0	1
01	0	1
10	0	0
11	0	0

_____

100001

11101110

АЛГОРИТМИ ПЕРЕДБАЧЕНЬ

Передбачення для циклів з відомою кількістю виконання.

Окремо для циклів (n < 64 – 6 бітів)

- Передбачення для непрямих переходів – одна адреса переходу
- Передбачення для fun … return

УНИКНЕННЯ ПЕРЕХОДІВ В РАЗІ НАЯВНОСТІ ПЕРІОДУ

1 Обчислити суми елементів масиву з парними та непарними номерами

2 Запрограмувати формулу:

y[i]=f1(x[i]) i = 0, 3, 6,…3k

y[i]=f2(x[i]) i = 1, 4, 7,…3k+1

y[i]=f3(x[i]) i = 2, 5, 8,…3k+2

3 Функція має загальний вид:

```
… fun (….)
{
        // Check parameters
        ...
        // Fun Code
        ….
}
```

Обрати найбільш ефективний код

УНИКНЕННЯ ПЕРЕХОДІВ В РАЗІ НАЯВНОСТІ ПЕРІОДУ

Обчислити суми елементів масиву з парними та непарними номерами

```
void sum1 (int *x, int n, int&s1, int &s2){
        s1 = 0;  s2 = 0;
        for (int i = 0; i < N; i++){
                if (i%2 == 0)  s1+= x[i]; else s2+= x[i];
        }
}
```

Time1 = 0.2, Time2 = 0.08 (N =100000000)

хай необхідно запрограмувати формулу:

y[i]=f1(x[i]) i = 0, 3, 6,…3k

y[i]=f2(x[i]) i = 1, 4, 7,…3k+1

y[i]=f3(x[i]) i = 2, 5, 8,…3k+2

1 варіант

```
void fun33_1 (int *x, int *y, int n){
for (int i = 0, k = 0; i < n; ++i, k++){
if (i == 3 * k)  y [i] = fun1 (x[i]); else{if (i == 3 * k + 1) y [i] = fun2 (x[i]); else y [i] = fun3 (x[i]);}}}
```

Time1 = 1.14, Time2 = 0.38 (N =100000000)

ТИПИ ПАРАЛЕЛІЗМУ. ПАРАЛЕЛІЗМ НА РІВНІ КОМАНД. СПЕКУЛЯТИВНЕ ВИКОНАННЯ

Є 2 конвеєри.
Для команд умовного переходу виконується обидві гілки.
Усі отримані результати записуються в тимчасову пам’ять

Після того. як значення умови відомо, з обох варіантів обирається потрібний.
Тимчасові дані правильного варіанту записуються в пам’ять для результатів.

Є тільки в дорогих процесорах!
Перевага – практично немає витрат, пов'язаних з переходами.
Недоліки .
Можливо тільки для команд, інформація для яких в БПП.
Ускладнення структури процесору.
Витрати ресурсів.

ТИПИ ПАРАЛЕЛІЗМУ. ПАРАЛЕЛІЗМ НА РІВНІ КОМАНД. СУПЕРСКАЛЯРНІСТЬ

Паралелізм на рівні команд. Суперскалярність Та Довгі команди

Що таке суперскалярний процесор?

Є декілька блоків для виконання операцій або навіть декілька конвеєрів

Залежні та незалежні команди

X = a;	**X= a;**	**X= a;**	**y = c;**
z = x + b;	**z = x + b; y = c;**	**z = x + b;**	
y = c;	**u = z + y;**	**u = z + y;**	
u = z + y;			

2 режими обробки:

Процесор перевіряє наявність залежностей і виконує паралельно ті команди, які незалежні – перевірка динамічн[illegible]ий процесор

Компілятор для конкретних типів процесорів (з урахуванням наявності паралельних блоків) під час компіляції в одну довгу команду з'єднує ті операції. Які можна виконувати паралельно - (VLIW команди – Very Long Instruction Word)

ТИПИ ПАРАЛЕЛІЗМУ. ПАРАЛЕЛІЗМ НА РІВНІ КОМАНД. СУПЕРСКАЛЯРНІСТЬ

Приклад використання суперскалярності

- **Хай необхідно обчислити значення поліному:**

$Y = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$

Метод Горнера

$Y = (\ldots((a_n x + a_{n-1}) x + a_{n-2})x \ldots + a_1)x + a_0$

При n = 3 маємо:

$Y = a_3 x^3 + a_2 x^2 + a_1 x + a0$ и $Y = ((a_3 x + a_2) x + a_1)x + a_0$.

По методу Горнера необхідно послідовно виконати 3 операції множення та 3 операції додавання незалежно від кількості пристроїв для виконання цих операцій, всього 6 операцій.

Схема обчислення для процесора з 3 пристроями для цілих чисел:

$a_3 x$	$a_2 x$	$a_1 x$
$a_3 x^2$	$a_2 x^2$	$a_1 x + a_0$
$a_3 x^3$	$a_2 x^2 + a_1 x + a_0$	
$a_3 x^3 + a_2 x^2 + a_1 x + a_0$	Всього 4 операції	

ТИПИ ПАРАЛЕЛІЗМУ. ПАРАЛЕЛІЗМ НА РІВНІ ДАНИХ. SIMD КОМАНДИ

Паралелізм на рівні даних. SIMD команди

Класи команд:

- **MMX** (64 біт, 2, 4, 8 елементів, цілі, регістри з плаваючою точкою).
- **3DNow!** (64 біт, 2 float, регістры з плаваючою точкою, горізонтальні операції)
- **SSE** (128 біт, int, double, float, 2-16 елементів, розширений набір операцій)
- **AVX** (256 (512, 1024))

ТИПИ ПАРАЛЕЛІЗМУ. ПАРАЛЕЛІЗМ НА РІВНІ ЗАДАЧ

Паралелізм на рівні задач

Потоки **(**визначення**)**

Способи завдання паралелізму на рівні задач

- **Псевдо** паралелізм **(**операційна система, витіснення**)**
- **HYPER Threading (**операційна система, витіснення, якщо кількість потоків більше ніж 2, додатковий набір регістрів та PIC**)**
- **multi-core (**операційна система, витіснення, якщо кількість потоків більше кількості ядер**)**
- **many-core** (операційна система, спеціальні бібліотеки, наприклад, CUDA технологія)
- **Системи з розподіленою пам'яттю** (Системи з масовим паралелізмом або кластери) – розподілені операційні системи

СТРУКТУРА БАГАТОЯДЕРНОГО ПРОЦЕСОРУ

Процесорне ядро 1 (Кеш L1)	Процесорне ядро 2 (Кеш L1)	Процесорне ядро 3 (Кеш L1)	Процесорне ядро 4 (Кеш L1)
Кеш L2	Кеш L2	Кеш L2	Кеш L2
Кеш L3 (2 / 4 Мбайт)			
Інтерфейс системних запитів			
Перехресний комутатор			

ВИСНОВКИ

- Чому сьогодні треба вчитися розробляти програми з паралельними обчисленнями? – апаратні засоби працюють ефективно тільки в разі таких обчислень!
- Чи задовольняють сучасні ОС вимогам паралельного програмування? – не зовсім, плохо вміють розподіляти навантаження, валики накладні витрати, .повязані з використанням потоків?
- Чи достатньо перекомпілювати програму для отримання паралельних обчислень? Ні. Сучасні компілятори вміють забезпечити паралельне виконання тільки циклів, і то в разі, якщо гарантується незалежність ітерацій циклу.
- Чи довільний алгоритм можна виконувати паралельно. Ні, навіть найпростіші алгоритми створені з урахуванням послідовного обчислення – треба вчитися розробляти спеціальні алгоритми.
- Чи є мова програмування для створення паралельних програм? – Такі мови є (функціонального програмування), але вони не ефективні для систем з загальною пам'яттю. Сьогодні створюються нові мови. Приклад – мова Axum
- Чи є технології для розробки паралельних програм. Так, і їх вивчення є задачею цієї дисципліни

ПИТАННЯ ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ

- Багатопроцесорні обчислювальні і операційні системи.
- Обчислювальні системи з загальною та розподіленою пам'яттю.

Паралельне програмування (учбовий посібник)

МАТЕРІАЛИ ДЛЯ ЕКСПРЕС-КОНТРОЛЮ

- Назвіть рівні паралелізму
- До якого рівня паралелізму відноситься конвеєрна обробка команд?
- Виведіть формулу залежності часу виконання *n* команд для конвеєра з *m* вузлами, якщо час виконання однієї команди у випадку відсутності конвеєра дорівнює одиниці. Накладними витратами, пов'язаними з використанням конвеєра, можна зневажити. Зробіть висновки про залежності цього часу від кількості вузлів конвеєра.
- Що таке статичне й динамічне прогнозування переходів, як їх варто враховувати при складанні програм?
- Проаналізуйте код, що формується транслятором для операторів for, while, do, switch і сформулюйте рекомендації з їхнього використання.
- Складіть функцію обчислення найбільшого загального дільника й мінімізуйте кількість команд переходу для цієї функції.
- Напишіть код для обнуління молодших *n* і старших *m* біт в 32 бітному числі, не використовуючи переходів (*m, n* - константи, $m + n < 32$)
- Складіть функцію упорядкування даних методом простої вставки, з мінімальною кількістю команд переходів.
- Розгляньте класичний алгоритм обчислення суми й змініть його таким чином, щоб кількість необхідних тактів була мінімальною для заданої кількості ядер процесора.
- Чим відрізняються системи з розподіленою й загальною пам'яттю, до якого типу систем відносяться системи на основі багатоядерного процесору