використання властивостей сучасних процесорів при створенні програм.

Конвеєр

Мета заняття

Як показує практика оптимізації програм обчислювальна складність визначається не тільки кількістю операцій, які треба виконати, але і врахуванням властивостей обчислювальної системи. Навчитися створювати найбільш ефективні програми з урахуванням конвеєру сучасних процесорів – мета заняття

Методичні вказівки з організації самостійної роботи студентів

Вивчить властивості конвеєрів сучасних процесорів. Аналіз цих властивостей показує, що кількість команд переходу в програмі треба мінімізувати. Особливу увагу необхідно надати передбаченню переходів [1, с. 18-34].
Передбачення переходів (з'явилося, починаючи з Pentium) дозволяє зменшити витрати продуктивності конвеєра у зв'язку з наявністю переходів.
Використовується статичне і динамічне передбачення.

Статичне передбачення

Статичне передбачення використовується для команд переходу, які виконуються в перший раз. Статичне передбачення для Pentium – є виконання наступної команди, тобто фактично немає передбачення.
Для більш досконалих процесорів для посилань вперед вважалося, що переходу не буде, для посилань назад, що перехід буде. Це пов'язано з тим, що посилання назад відповідає циклу, а цикли пишуться для того, щоб часто виконувати одну й ту саму ділянку коду.
У процесі статичного передбачення команда переходу заноситься в буфер передбачення переходу разом з адресою, куди фактично виконався перехід. Крім цього встановлюється прапор в 1, якщо перехід фактично відбувся.

Динамічне передбачення

Використовується для команд переходу, які виконуються повторно і для яких вже є інформація. Для кожної команди переходу записується її адреса (Адрес1), адресу, куди необхідно перейти (Адрес2) і додаткова інформація, за допомогою якої враховується історія переходів. Найбільш просто враховується період, якщо він є, наприклад, коли наявність переходу і його відсутність упорядковані. Для сучасних процесорів може враховуватись кількість виконання циклів, та команди виклику функцій та повернення після їх виконання. Як і довільне передбачення, воно може бути невірним, тому рекомендується оптимізувати кількість переходів. Треба враховувати кінцевий розмір блоку прогнозування, він може зберігати інформацію тільки про невелику кількість команд переходу, тому якщо в циклі багато команд переходу, є імовірність, що для наступної ітерації інформація про команди переходу буде замінена інформацією про останні команди переходу ітерації.
Огляд методів передбачення переходів показує, що втрати в продуктивності завжди будуть при наявності команд переходу, тому необхідно мінімізувати кількість команд переходу в програмі!

Контрольні запитання і завдання

1 Що таке статичне та динамічне прогнозування. Наведіть приклади.
2 Чому не періодичні переходи приводять до максимальних витрат продуктивності процесору?
3 Проаналізуйте код, який формується при реалізації циклів типу for, while, do, сформуйте рекомендації по їх застосуванню.
4 Проаналізуйте код, який формується за допомогою оператору switch, та час виконання коду з використанням switch та заміною його еквівалентними операторами if. Сформуйте рекомендації по їх застосуванню.

Приклади аудиторних задач

Приклад 1.
Задано масив чисел. Числа, які стоять на парних містах додавати, а числа, які стоять на непарних місцях – віднімати, тобто знайти s = x[0]-x[1]+ x[2]- x[3]….
Вимоги до функції: вона не повинна залежати від типу даних
Перевірити наступні варіанти:
Використання індексних змінних;
Використання показників;
Варіант 1

```
template <typename T>
T  Example1_1 (T *x, size_t n){
        T s =0;
                for (size_t  i = 0; i < n; ++i){
                        if (i%2 == 0)
                                s+=x [i];
else s-=x[i];
                }
        return s;
}
```

Варіант 2

```
template <typename T>
T  Example1_2 (T *x, size_t n){
        T s =0;
        T*px = x;
                for (size_t  i = 0; i < n; ++i){
                        if (i%2 == 0)
                                s+=*px++; else s-=*px++;
                }
        return s;
}
```

Варіант 3 (з індексними змінними та показниками)
Видалити порівняння в циклі

```
template <typename T>
T  OptExample1_1 (T *x, size_t n){
        T s =0;
```

```
		for (size_t  i = 0; i < n; i +=2){
				s+=x [i] - x [i+1];
		}
	return s;
}

template <typename T>
T  OptExample1_2 (T *x, size_t n){
	T s =0;
	T* px = x;
		for (size_t  i = 0; i < n; i +=2){
				s+=*px++;
				s-=*px++;
		}
	return s;
}
```

Таблиця результатів

	Example1_1	Example1_2	OptExample1_1	OptExample1_1
Time (float)	0.023	0.023	0.011	0.011

Висновки.
Використання показників і індексів приводить практично до однакових результатів
Видалення перевірок суттєво впливає на продуктивність

Приклад 2.
Знайти кількість негативних чисел масиву цілих чисел
Розглянути 3 варіанти:
1 усі числа позитивні.
2 Усі числа негативні
3 Випадкові числа
Для типу чисел обрати цілі типи.
Варіант 1

```
size_t Example2_1 (int *x, size_t n){
	size_t count = 0;
	for (size_t i = 0; i < n; i++)    {
		if (x [i] < 0)count++;
	}
	return count;
}
```

Варіант 2

```
size_t OptExample2_1 (int *x, size_t n){
	size_t count = n;
	for (size_t i = 0; i < n; i++){
			count += (x [i]>> 31);
	}
	return count;
}
```

Результати:

Функція	Example2_1			OptExample2_1		
Варіанти	+	-	+-	+	-	+-
Час	0.0098	0.0054	0.0049	0.0039	0.0039	0.0039

Висновки.

1 Всупереч очікуванню продуктивність найвища для усіх негативних чисел, а найменша для чисел з випадковими знаками.

2 Видалення переходів підвищує ефективність дл усіх варіантів вхідних даних. Ця продуктивність становиться вище, ніж найкращий варіант з перевіркою.

Приклад 3.

Виконати упорядкування методом пухирька для цілих чисел

Варіант 1. Звичайний метод.

Варіант 2 Кожне наступне число порівнювати з попередніми

```
void sort(int *x, int n){
	for (int p = n - 1; p >= 1; p--){
		for (int i = 0; i < p; i++)		{
			if (x[i] > x[i + 1]){
				int r = x[i]; x[i] = x[i + 1]; x[i + 1] = r;
			}
		}
	}
}
void OptSort(int *x, int n){
	for (int p = 1; p < n; p++){
		for (int i = p-1; i >=0; i--){
			if (x[i] > x[i + 1]){
				int r = x[i]; x[i] = x[i + 1]; x[i + 1] = r;
			}
		}
	}
}
```

Результати:

Варіант 1: 20

Варіант 2: 7

Висновок.

Якщо мінімізувати кількість помилок прогнозування, продуктивність збільшилась практично в 3 рази.

Приклад 4.

Задано 2 полінома. Перший має цілі коефіцієнти, другий має значення коефіцієнтів 0, 1, -1. Імовірність усіх значень однакова.

Обчислити добуток цих поліномів

```
void mulpol(int *x, int *y, int *z, int n) {
memset(z, 0, 2 * n * sizeof(int));
for (int i = 0; i < n; i++)
for (int j = 0; j < n; j++)
z[i + j] += x[i] * y[j];
}

void mulpolOpt1(int *x, int *y, int *z, int n) {
memset(z, 0, 2 * n * sizeof(int));
for (int j = 0; j < n; j++)
if (y[j]){
for (int i = 0; i < n; i++)
```

```
z[i + j] += x[i] * y[j];
}
}

void mulpolOpt2(int *x, int *y, int *z, int n) {
memset(z, 0, 2 * n * sizeof(int));
for (int j = 0; j < n; j++) {
if (y[j] != 0) {
if (y[j] == 1) {
for (int i = 0; i < n; i++) z[i + j] += x[i];
}
else{
for (int i = 0; i < n; i++) z[i + j] -= x[i];
}
}
}
}

void mulpolOpt3(int *x, int *y, int *z, int n) {
memset(z, 0, 2 * n * sizeof(int));
for (int j = 0; j < n; j++) {
if (y[j] != 0) {
int *pz = z + j;
if (y[j] == 1) {
for (int i = 0; i < n; i++)  pz[i ] += x[i];
}
else {
for (int i = 0; i < n; i++) pz[i ] -= x[i];
}
}
}
}
```

Результати

Функція	mulpol	mulpolOpt1	mulpolOpt2	mulpolOpt3
Час	0.075	0.071	0.049	0.02

Висновок. Врахування значень другого поліному та перед обчислення індексу дозволило прискорити обчислення більше ніж в 3 рази

Приклади домашніх задач

1 Скласти функцію для обчислення суми ряду $\sum(-1)^n x^n/n!$

2 Скласти функції, які округляє задане ціле число на цілу границю (число вважається вирівняним, якщо воно ділиться націло на задану границю) в більшу (High) та меншу (Low) сторони.

3 Скласти функцію, яка округляє задане число до найближчого цілого