Змістовний модуль:

СУЧАСНІ ПРОЦЕСОРИ ТА ПАРАЛЕЛІЗАЦІЯ ОБЧИСЛЕНЬ

Розділ: КРИТЕРІЇ ТА ПОКАЗНИКИ ПРОГРАМ

Лекція 4. КРИТЕРІЇ ТА ПОКАЗНИКИ ПРОГРАМ. ЕКСПЕРИМЕНТАЛЬНІ МЕТОДИ

ПИТАННЯ ДЛЯ ВИВЧЕННЯ

1. Повторення аналітичних методів
2. Класифікація функцій для виміру часу.
3. Стандартні функції мови С.
4. Використання функцій *WinApi* для визначення часу.
5. Використання лічильника тактів *TSC* центрального процесора без та з допомогою Intrisinc засобів.
6. Особливості виміру часу для багато потокових додатків.
7. Використання функцій виміру часу для середовищ розробки паралельних додатків.
8. Використання внутрішніх методів профілювання для виміру часу виконання коду.

ПРИКЛАД

Визначити прискорення, ефективність та вартість для обчислення значення багаточлена:

$Pn(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_i x^i + \ldots + a_1 x^1 + a_0$.

Необмежений паралелізм. n процесорів.

Послідовний алгоритм.

Ефективний послідовний алгоритм (схема Горнера) вимагає для обчислення 2*n операцій. Припустимо, що час обчислень пропорційний кількості операцій. Тоді значення показників :

$Sp(n) = 2*n /(2*n) = 1$

$Ep(n) = 1$

$Cp(n) = (T(p) *p) = 2 * n.$

ПРИКЛАД

Паралельний алгоритм 1.

Кількість процесорів дорівнює *n*.

a_nx	$a_{n-1}x$	…	a_1x	*(n процесорів)*
a_nx^2	$a_{n-1}x^2$	…	$a_1x + a_0$	*(n процесорів)*
a_nx^3	$a_{n-1}x^3$	…	$a_2x^2 + a_1x + a_0$	*(n - 1 процесорів)*
…				
a_nx^n	$a_{n-1}x^{n-1} + \ldots + a_0$			*(2процесора)*
Y				
				(1 процесор)

Треба *n + 1* кроків і *n* процесорів для обчислення значення багаточлена:

$Sp(n) = 2*n /(n + 1)$

$Ep(n) = 2/(n + 1)$

$Cp(n) = n*(n + 1)$

Недолік: Нерівномірний розподіл процесорів між окремими кроками.

ПРИКЛАД

Паралельне обчислення. Алгоритм 2.

x^2		$r = a_1x$	*2 процесора*
x^3	a_2x^2	$r \mathrel{+}= a_0$	*3 процесора*
x^4	a_3x^3	$r \mathrel{+}= a_2x^2$	
x^5	a_4x^4	$r \mathrel{+}= a_3x^3$	
…			
x^n	$a_{n-1}x^{n-1}$	$r \mathrel{+}= a_{n-2}x^{n-2}$	
	a_nx^n	$r \mathrel{+}= a_{n-1}x^{n-2}$	
		$r \mathrel{+}= a_nx^n$	

Треба *n* + *1* кроків і 3 процесори для обчислення значення багаточлена:

*Sp(n) = 2*n /(n + 1)*

*Ep(n) = 2/3*n /(n + 1)*

*Cp(n) = 3**(n + 1)

ПРИКЛАД

Порівняння алгоритмів для обчислення значення поліному

- Прискорення за рахунок паралельного виконання однакові для обох алгоритмів, але при цьому Алгоритм 2 більш ефективний і його вартість набагато менше, ніж для Алгоритму 1.
- при збільшенні кількості ядер (n > 3) прискорення для алгоритму 2 не змінюється, а показники *Ep(n)* і *Cp(n)* погіршуються, але при цьому залишаються не гірше, ніж для Алгоритму 1;
- алгоритм 2 потребує накопичення суми, що може погіршите його характеристики.

А чи можна побудувати алгоритм, ефективність якого визначається кількістю процесорів?

Спробуємо!!!

ПРИКЛАД

Алгоритм 3 обчислення багаточлену.

Нехай у нас є *p* ядерний процесор (*p* < *n*). Нехай для простоти *n* кратне *p-1*. Якщо це не так, то старші коефіцієнти можна доповнити нулями.

Розділимо багаточлен на *m* = *p-1* порцій однакового розміру. У кожну порцію входять суміжні елементи багаточлена, розмір кожної порції дорівнює

k = n / (p - 1). p = 5

$Pn(x) = a_{99}x^{99} + a_{98}x^{98} + \ldots + a_1x^1 + a_0$

Представимо наш багаточлен у вигляді:

$Pn(x) = A_{m-1}x^{(m-1)k} + A_{m-2}x^{(m-2)k} + \ldots + A_1x^k + A_0$, $Pn(x) = A_3x^{75} + A_2x^{50} + A_1x^{25} + A_0$

де:

m = p – 1 (кількість порцій) *4*,

k = n / (p - 1) – Кількість елементів в порції *25*

$A_0 = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \ldots + a_1x^1 + a_0$, $A_0 = a_{24}x^{24} + a_{23}x^{23} + \ldots + a_1x^1 + a_0$,

$A_1 = a_{2k-1}x^{k-1} + a_{2k-2}x^{k-2} + \ldots + a_{k+1}x^1 + a_k$, $A_1 = a_{49}x^{24} + a_{48}x^{23} + \ldots + a_{26}x^1 + a_{25}$,

... ...

$A_{m-1} = a_{mk-1}x^{k-1} + a_{mk-2}x^{k-2} + \ldots + a_{(m-1)k+1}x^1 + a_{(m-1)k}$ $A_3 = a_{99}x^{99} + a_{98}x^{98} + \ldots + a_{76}x^1 + a_{75}$,

$A_0, A_1, \ldots A_{m-1}$ - «цифри» в системі числення, рівної x^k.

ПРИКЛАД

Схема паралельних обчислень для алгоритму 3:

Крок 1	Крок 2	Крок 3	Крок 4
P1: A_0	$x^m, x^{2m},\ldots$		$\sum$
P2: A_1	$x^{(p-1)m}$	$A_1 * x^k$	
P3: A_2.		$A_2 * x^{2k}$	
…		…	
Pp: A_{m-1},		$A_{m-1} * x^{(m-1)k}$	
2k	$log_2 m+p$	*1*	$log_2 p$

ПРИКЛАД

Показник (n =1000, p=4)	Схема Горнера	Алгоритм 1	Алгоритм 2	Алгоритм 3
Sp(n)	1	2n/(n+1) 1.998	2n/(n+1) 1.998	2n/(2k+ $\log_2 2n + p$) 3.88
Ep(n)	1	2/(n+1) 0. 001996	2n/(3(n+1)) 0.666	2n/(p(2k+ $\log_2 2n + p$)) 0.97
Cp(n)	2n 2000	n(n+1) 1001000	3(n+1) 3003	p(2k+ $\log_2 2n + p$)) 2060

НЕДОЛІКИ АНАЛІТИЧНИХ МЕТОДІВ

1. Виведення формули обчислення кількості операцій може бути дуже складним, особливо з урахуванням різних варіантів виконання програми (різні гілки програми мають різну обчислювальну складність).
2. Не враховує суперскалярну архітектуру процесора.
3. Не враховує складність операцій, так операція додавання й множення має різну складність, а враховуються у формулі як однакові.
4. Не враховує накладних витрат, пов'язаних з паралельним виконанням, а ці витрати можуть бути істотними, якщо паралельні гілки мають невелику обчислювальну складність.

ТЕХНОЛОГІЯ ВИМІРУ ЧАСУ

1. Виміри треба виконувати на реальній, а не на віртуальній машині.
2. Встановити режим Release для додатку.
3. Припинити виконання усіх інших додатків
4. Вибір функцій для виміру часу визначається потрібною точністю вимірів
5. Для зменшення впливу переключень вимір часу виконувати багатократно. З усіх вимірів обирати мінімальний час.
6. Необхідно забезпечити неможливість визначення обчислення функції зайвим оператором

ТЕХНОЛОГІЯ ВИМІРУ ЧАСУ

Структура коду для вимірів:

```
Type MinTime = MAXVALUE;
for (int i = 0; i < n; ++i){
        Type Start =…
        Code
        Type Finish = …
        if (Finish – Start < MinTime)
                MinTime = Finish – Start;
}
```

Приклад. Виміряти час для функції обчислення квадратів числа (clock)

```
typedef long clock_t;
clock_t clock(void);
#define CLOCKS_PER_SEC  1000
```

ТЕХНОЛОГІЯ ВИМІРУ ЧАСУ

Структура коду для вимірів:

```
Type MinTime = MAXVALUE;
for (int i = 0; i < n; ++i){
        Type Start =…
        Code
        Type Finish = …
        if (Finish – Start < MinTime)
                MinTime = Finish – Start;
}
```

Приклад. Виміряти час для функції обчислення квадратів числа (clock)

```
typedef long clock_t;
clock_t clock(void);
#define CLOCKS_PER_SEC  1000
```

Приклад.

```
void Sq(int a[], int b[], size_t n){
    for (size_t i = 0; i < n; i++)
            b[i] = a[i] * a[i];
}
…
clock_t cMin = 0x7FFFFFFF;
clock_t start, finish;
for (size_t i = 0; i < 10; i++){
start=clock();Sq(a, b, N);finish = clock();
if (finish - start < cMin)
    cMin = finish - start;
} printf("time = %lg", (cMin + 0.) /
    CLOCKS_PER_SEC);
```

time =0.022c (22 mc)

ВИЗНАЧЕННЯ ТОЧНОСТІ ОБЧИСЛЕННЯ

Чому точність обмежена?

Кварцевий генератор, частота 1193180 раз в секунду.

Тік таймера 1193180/65536 = 18.2 рази в секунду (було раніше)

Як сьогодні?

NTSTATUS NtQueryTimerResolution(
OUT PULONG *MinimumResolution*,
OUT PULONG *MaximumResolution*,
OUT PULONG *CurrentResolution*);

Як звернутися до не документованої функції?

1. Впізнати, в якій DLL визначена.
2. Звернутися до функції DLL.
3. Ця функція визначена в DLL ntdll.dll
4. Typedef

- HMODULE WINAPI GetModuleHandleW(_ LPCWSTR lpModuleName);
- FARPROC WINAPI GetProcAddress(HMODULE hModule, _In_ LPCSTR pProcName);

ВСТАНОВЛЕННЯ ТОЧНОСТІ ОБЧИСЛЕННЯ

```
typedef INT(WINAPI *tNtQueryTimerResolution)(
        PULONG, PULONG, PULONG);
ULONG Minimum, Maximum, Current;
HMODULE hDLL = GetModuleHandle(TEXT("ntdll.dll"));
if (hDLL){
    tNtQueryTimerResolution NtQueryTimerResolution =
    (tNtQueryTimerResolution)GetProcAddress(hDLL,
        "NtQueryTimerResolution");
    if (NtQueryTimerResolution){
        NtQueryTimerResolution(&Minimum, &Maximum, &Current);
        printf("Minimum = %d Maximum = %d Current = %d\n", Minimum,
        Maximum, Current);
    }
}
```

W 8 Minimum 156250, maximum 5000, Current = 10006 (100 нс)

ВСТАНОВЛЕННЯ ТОЧНОСТІ ОБЧИСЛЕННЯ

Алгоритм визначення точності виміру часу.

1 Зробити поточний вимір часу.

2 в циклі виконувати вимір часу до тих пір, поки він співпадає з попереднім

3 різниця вимірів часу і є точність виміру.

Приклад. Визначити точність виміру для функції clock ()

```
typedef long clock_t;
clock_t clock(void);
#define CLOCKS_PER_SEC  1000
```

ВСТАНОВЛЕННЯ ТОЧНОСТІ ОБЧИСЛЕННЯ

Алгоритм визначення точності виміру часу.

1 Зробити поточний вимір часу.

2 в циклі виконувати вимір часу до тих пір, поки він співпадає з попереднім

3 різниця вимірів часу і є точність виміру.

4 повторити вимір декілька разів

Приклад. Визначити точність виміру для функції clock ()

```
double GetAccuracyClock(){
    clock_t Start, Finish;
    Start = clock();
    do    {
          Finish = clock();
    } while (Finish - Start == 0);
    return (Finish - Start + 0.) / CLOCKS_PER_SEC;
}
```

Результат : 0.001 с (1 мс)

КЛАСИФІКАЦІЯ ЗАСОБІВ ВИМІРУ ЧАСУ

1. **Лічильник тактів центрального процесора Time-Stamp Counter (TSC).**
2. Функції ОС.
3. Стандартні функції мови С, С++.
4. Засоби виміру часу в середовищах розробки паралельних програм.
5. Засоби профілювання

ВИКОРИСТАННЯ РАХІВНИКА ТАКТІВ

Підтримка RDTSC

```
inline BOOL IsRDTSCSupport (){
        _asm    {
                mov     eax, 1
                cpuid
                mov     eax, 1
                test    edx, 10000B
                jne     short m1
                sub     eax, eax
                m1:


        }
}
```

Вимір часу

```
inline unsigned __int64 GetTacts (){
        __asm{
                mov     eax, 0
                rdtsc
        }
}
```

INTRINSIC ФУНКЦІЇ

Альтернатива асемблерним вставкам.

З'явилися, починаючи з VS 2005.

Для використання необхідно:

- підключити заголовний файл *intrin.h*.
- для визначення кількості тактів процесора використовувати функцію *__rdtsc ()*.

INTRINSIC ФУНКЦІЇ

```
inline unsigned __int64 GetTacts (){
        #ifdef __INTRIN_H_
                return __rdtsc ();
        #else
        __asm rdtsc
        #endif
}
```

Порівняння функцій з використанням та без використання INTRINSIC засобів показує, що час їх виконання однаковий!!!

Перевага INTRINSIC засобів: не треба використовувати асемблер.

ВИКОРИСТАННЯ РАХІВНИКА ТАКТІВ

Приклад. Визначити час виконання функції *GetTacts.*

```
unsigned __int64 Start, Finish, Dif, MinDif0=(unsigned __int64)(-1), MinDif = MinDif0;
// 10 спроб для виміру
for (int i = 0; i < 10; ++i) {
    Start = GetTacts (); Finish = GetTacts (); Dif = Finish - Start; if (MinDif0 >Dif)
    MinDif0 = Dif;
}
printf ("Time = %I64d\n", MinDif0);
```

Результат: time (*GetTacts*) = 63; Time (додавання) = 1080.

ВИКОРИСТАННЯ РАХІВНИКА ТАКТІВ

Переваги:

- точність виміру максимальна (у тактах процесора);

Похибка не більше 63 тактів

- Немає накладних витрат, пов'язаних з визначенням часу

Недоліки:

- у багатоядерному процесорі кожне ядро має свій лічильник тактів. Для визначення загального часу виконання не можна порівнювати виміри, зроблені в різних потоках;
- не можна використовувати при порівнянні алгоритмів на різних комп'ютерах, тому що тривалість такту залежить від тактової частоти;
- не гарантується наявність цієї команди для процесорів з іншою архітектурою;

Рекомендація по використанню:

Використовувати **тільки** для виміру дуже невеликих інтервалів часу

Функції операційної системи

ФУНКЦІЯ *GetTickCount*

DWORD GetTickCount ();

Повертає кількість мілісекунд, що пройшли з моменту старту ОС дотепер.

Приклад. Визначити точність виміру часу за допомогою функції GetTickCount .

```
for (int i = 0; i < 10; i++)      {
    DWORD Start = GetTickCount (),
          Finish = GetTickCount ();
    while (Start == Finish)
          Finish = GetTickCount ();
    printf ("Start = %d Finish = %d Finish - Start =%d\n",
          Start, Finish, Finish - Start);}
```

Результат:

15 або 16 мс

Використовує мінімальну можливу точність - для узгодження зі старими версіями Windows (DOS)

ФУНКЦІЯ *GETSYSTEMTIMEASFILETIME*

void GetSystemTimeAsFileTime(LPFILETIME lpSystemTimeAsFileTime);

Визначає системний час, який ОС записує в пам'ять з певною частотою.
Час виміряється відносно 01.01.1601 року в одиницях часу 100 наносекунд.

Структура результату:

Структура FILETIME:

```
typedef struct _FILETIME {
 DWORD dwLowDateTime, dwHighDateTime;
}FILETIME, *PFILETIME;
```

Перетворення структури в 64-бітне число.

```
#define FILETIMETOUINT64(ft) \
((((unsigned __int64)ft.dwHighDateTime) << 32) | (ft.dwLowDateTime))
```

Накладні витрати, пов'язані з її використанням, мінімальні.

ФУНКЦІЯ *GETSYSTEMTIMEASFILETIME*

Приклад. Визначити частоту, з якої ОС обновляє системний час (точність виміру часу).

```
int _tmain(int argc, _TCHAR* argv[])
    {
        FILETIME ft, ft1;
        GetSystemTimeAsFileTime (&ft);
        while (1) {
            GetSystemTimeAsFileTime (&ft1);
            if (ft1.dwLowDateTime != ft.dwLowDateTime) break;
        }
        printf ("GetSystemTimeAsFileTime: %I64d\n",
        FILETIMETOUINT64(ft1) - FILETIMETOUINT64(ft));
        return 0;
}
```

Цикл виконується до тих пір, поки наступний замір співпадає з попереднім.
Результат: 10000 тиків. (10000 /10^7 = 0.001с (1 mc).

Точність виміру 1 мс (як clock)

ФУНКЦІЯ *GetSystemTimePreciseAsFileTime*

- Функцію додано, починаючи з Windows 8

Для перевірки можливості використання

```
#if defined(NTDDI_WIN8) && NTDDI_VERSION >= NTDDI_WIN8
GetSystemTimePreciseAsFileTime( &ftStart );
#else
GetSystemTimeAsFileTime( &ftStart );
#endif
```

Точність $1175/10^7$ .. $6900/10^7$ ($5000/10^7$) = 0.5 mc

ФУНКЦІЇ QueryPerformanceCounter ТА QueryPerformanceFrequency

Вимір тактової частоти процесору:

BOOL QueryPerformanceFrequency (LARGE_INTEGER*lpFrequency);

Вимір поточного часу:

BOOL QueryPerformanceCounter(LARGE_INTEGER*lpPerfCount);

```
typedef union _LARGE_INTEGER
{
    struct
    {
        DWORD  LowPart;
        LONG   HighPart;
    };
    struct {
        DWORD  LowPart;
        LONG   HighPart;
     } u;
     LONGLONG   QuadPart;
} LARGE_INTEGER, *PLARGE_INTEGER;
```

Точність: 4.27E-7

ФУНКЦІЇ QueryPerformanceCounter ТА QueryPerformanceFrequency

```
double GetAccuracyQuary(){
        LARGE_INTEGER liStart, liFinish;
        QueryPerformanceCounter(&liStart);
        do      {
                QueryPerformanceCounter(&liFinish);
        } while ((liFinish.QuadPart - liStart.QuadPart) == 0);
        LARGE_INTEGER liFreq;
        QueryPerformanceFrequency(&liFreq);
        return (double)(liFinish.QuadPart - liStart.QuadPart) / liFreq.QuadPart;
 }
```

Точність: 4.27E-7

Недолік функцій – функції або не дають потрібну точність, або складні в використанні

ПЕРЕВАГИ ТА НЕДОЛІКИ ФУНКЦІЙ ОС

Переваги

- Мінімальні накладні витрати
- Максимальна точність

Недоліки

- Залежать від платформи
- Складні в використанні

СТАНДАРТНІ ФУНКЦІЇ МОВ *C, C++*

time_t time (&time_t); _
_time32_t _time32(__time32_t *timer); __time64_t _time64(__time64_t *timer);
Функція повертає час в секундах, що пройшов з 1 січня 1970 року (32 або 64 біта).
Точність виміру: 1 с.
Приклад 2.6. Визначити максимальну дату, яку ще можна задати за допомогою 32-бітного значення time_t.

```
// Функція для перетворення часу з формату Time_t в формат
// SYSTEMTIME
VOID CvtTime_tToSYSTEMTIME (time_t told, PSYSTEMTIME pstTime)
{
    struct tm stm; localtime_s(&stm, &told); pstTime->wHour = stm.tm_hour;
    pstTime ->wMinute = stm.tm_min; pstTime ->wDay = stm.tm_mday;
    pstTime ->wMonth = stm.tm_mon + 1; pstTime ->wSecond = stm.tm_sec;
    pstTime->wMilliseconds = 0; pstTime->wDayOfWeek = stm.tm_wday;
    pstTime ->wYear = stm.tm_year + 1900;
}
…
time_t tMax = 0x7FFFFFFF; SYSTEMTIME st; CvtTime_tToSYSTEMTIME (tMax, &st);
```

Максимальна дата: 19 січня 2038 рік в 5 год. 14 хв. 07 сек.

СТАНДАРТНІ ФУНКЦІЇ МОВ *C, C++*

clock_t clock()

Измеряет время с момента запуска приложения.

clock_t – long (32 бита)

Это время в миллисекундах, для получения времени в секундах надо разделить на константу (CLOCKS_PER_SEC = 1000 **)**

Для проверки указанной точности можно использовать код:

```
clock_t  begin = clock (), end = begin;
while (begin == end)
{
end = clock ();
}
printf ("Clock: %g\n", (float)(end - begin)/CLOCKS_PER_SEC);
```

КЛАС chrono

```
#include <chrono>
using namespace std::chrono;
double GetChronoAccuracy(){
    auto start = high_resolution_clock::now();
    auto finish = high_resolution_clock::now();
    do{
        finish = high_resolution_clock::now();
    } while ((finish - start).count() == 0);
    auto dif = std::chrono::duration_cast<std::chrono::nanoseconds>
        (finish - start).count();
    return double (dif);
}
```

1E06 (нс) = 1 ms

Висновки по функціям мови:

Точність обчислення часу може бути не достатня для оцінки ефективності

ПЕРЕВАГИ ТА НЕДОЛІКИ ФУНКЦІЙ С (С++)

Переваги

- Не залежать від платформи
- Простота використання (clock)
- Можна виміряти великі діапазони

Недоліки

- Може бути недостатня точність

ВИКОРИСТАННЯ ФУНКЦІЙ ВИМІРУ ЧАСУ ДЛЯ СЕРЕДОВИЩ РОЗРОБКИ ПАРАЛЕЛЬНИХ ДОДАТКІВ

Функції виміру часу середовища розробки OPEN MP

Заголовчий файл: ***omp.h***.

Функції виміру часу:

double ***omp_get_wtime****(void);* // час (секунд)

double ***omp_get_wtick****(void);* // тривалість такту(секунд).

Кількість тактів = *omp_get_wtime()/ omp_get_wtick()*.

ВИКОРИСТАННЯ ФУНКЦІЙ ВИМІРУ ЧАСУ ДЛЯ СЕРЕДОВИЩ РОЗРОБКИ ПАРАЛЕЛЬНИХ ДОДАТКІВ

```
double GetOMPAccuracy(){
    double Start, Finish;
    Start = omp_get_wtime();
    do{
        Finish = omp_get_wtime();
    } while (Finish - Start == 0);
    return (Finish - Start );
}
```

Результат: Time = 4.2771e-007 sec

Аналіз функцій показує, що вони використовують функції Query…

ВІДНОСНИЙ ВИМІР ЧАСУ

Будемо використовувати для виміру часу функції, які вимірюють його з великою похибкою, наприклад, ***GetTickCount*** або ***clock***. Але будемо рахувати кількість циклів, яку можна виконати за заданий інтервал часу, наприклад, за дві секунди.

Код для виміру часу:

```
Count = 0;
Замір часу початку виконання
do
{
    // Вимірюваний код
    Замір часу закінчення виконання
    Count++;
}while (Кінцевий час - початковий час < 2000);
```

Яким чином можна доказати коректність такого вимірювання часу?

ВІДНОСНИЙ ВИМІР ЧАСУ

Методика перевірки коректності:

Для масивів з кількістю елементів N1, N2, N3 виміряємо час виконання заданної операції за допомогою функцій виміру часу T1, T2, T3.

Для тих же масивів вимірюємо час з використанням відносного виміру. Отримуємо T1’ = T/Count[0], T2’=T/Count[1], T3’=T/Count[2] (T – час для підрахунку).

Якщо T2/T1≈ T2’/T1’ та T3/T2≈ T3’/T2’, то такий вимір допустимий, інакше – ні.

ВІДНОСНИЙ ВИМІР ЧАСУ

Приклад.

```
DWORD dwStart, dwFinish;
    DWORD dwCount [VariantCount] = {0, 0, 0};
    sMax [0] = sMax [1] = sMax [2] = 0;
    for (int k = 0; k < VariantCount; ++k)        {
            dwStart = GetTickCount ();
            do{
                    s [k]= Summa (x, Sizes [k]);
                    dwFinish = GetTickCount ();
                    dwCount [k] +=1;
                    if (s [k] > sMax [k]) sMax [k] = s [k];
            }while (dwFinish - dwStart < 2000);
            printf ("sMax [k] = %d\n", sMax [k]);
    }
```

ПРОФІЛЮВАННЯ

Профілювання для VS2013

Для використання засобу профілювання необхідно виконати наступні кроки.

1. Створити exe файл для режиму *Release*. Нехай ім'я проекту *Performance*.
2. У головному меню VS2013 вибрати пункт *Analyze* (Аналіз).
3. У меню, що випадає, вибрати пункт*: Performance* and Diagnostics (запуск компонентів для аналізу продуктивності).
4. Обрати з Available Tools CPU Usage (використання процесору).
5. На запит про можливості змін для компютеру відповісти Так

В результаті отримуємо затрати часу (в %) для найбільш затратних функцій

ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ФУНКЦІЙ ВИМІРУ ЧАСУ

	C(C++)	WinApi	Intrinsic
Простота	+	-	-
Точність	-	+	+
Кросплатфор-меність	+	-	-
Можливість використання для багатопоточних додатків	+	+	-

НЕДОЛІКИ ЕКСПЕРИМЕНТАЛЬНИХ МЕТОДІВ

1. Вимагає попередньої програмної реалізації різних методів, що може вимагати багато часу.
2. Експериментальні результати можуть бути отримані тільки для наявних обчислювальних систем.
3. Вплив інших програм, в тому числі модулів операційної системи, може бути суттєвим.
4. Точність вимірювання обмежена точністю функцій, які використовуються. Час виконання самих функцій вимірювання може бути одного порядку з часом, що вимірюється.

ВИСНОВКИ

1. Експериментальні методи дозволяють визначити обчислювальну складність алгоритму з урахуванням усіх накладних витрат.
2. Є багато різних функцій і методів обчислення часу. Їх вибір визначається необхідною точністю, вимогам до кросплатформеності та типу: однопоточний або многопоточний.
3. Методи профілювання дозволяють визначити відносні часові затрати функцій програми. При цьому додавати в код функцій вимірювання часу не потрібно. Дозволять в програмі визначити функції, виконання який потребує більшу частину часу, тобто частину програми, яку треба оптимізувати в першу чергу. Насамперед, забезпечити її паралельне виконання.
4. Аналіз аналітичних та експериментальних методів показує, що обидва типа методів мають свої недоліки. Тому на практиці зазвичай за допомогою аналітичних методів обирається декілька найбільш ефективних алгоритмів, для яких за допомогою експериментальних методів після їх реалізації вимірюються реальні характеристики
5. Далі будуть розглянуті методи розробки програм з паралельними обчисленнями

ПИТАННЯ ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ

1. Особливості використання VTune для аналізу продуктивності програми

http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/

2. Засоби профілювання в Parallel Studio

http://visualstudiogallery.msdn.microsoft.com/d61ebebe-6cdf-4693-8cfa-f498c69adfec

МАТЕРІАЛИ ДЛЯ ЕКСПРЕС-КОНТРОЛЮ

1. Що таке часова складність алгоритму?

2. Порівняйте всі відомі Вам функції виміру часу по точності виміру. Запишіть ці функції в порядку збільшення точності обчислень.

3. Реалізуйте функцію ***GetTickCount*** за допомогою функції ***GetSystemTimeAsFileTime***.

4. Реалізуйте функції ***omp_gewtime(), omp_get_wtick()*** за допомогою функцій ***QueryPerformanceFrequency*** та ***QueryPerformanceCounter.***

5. Перевірте повторюваність результатів виміру часу для того самого коду. Поясніть отримані результати й сформулюйте рекомендації з виміру інтервалів часу