УДК 519.683.8

Саманцов О.О., Качко О.Г.

[1] *студент 3 курсу спеціальності програмна інженерія, ХНУРЕ*

[2] *професор кафедри програмного забезпечення ЕОМ, канд.тех.наук, ХНУРЕ*

e-mail: samantsov.aa@gmail.com

ПОРІВНЯННЯ МЕТОДІВ ПАРАЛЕЛІЗАЦІЇ ПРОГРАМ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ Windows Thread, OpenMP, Intel Thread Building Blocks

У роботі розглядаються основні методи паралелізації програм, особливості використання, порівняння часових характеристик програм написаних за допомогою цих методів.

Ключові слова: *паралелізація, Widows Thread, OpenMP, Intel Thread Building Blocks.*

Вступ

Останні роки пов'язані з різкою зміною напрямку розвитку процесорів – появою багатоядерних та багато поточних процесорів. Їх ефективне використання потребує загального переходу з послідовних програм на паралельні.

Існують дві основні моделі побудови паралельних програм. Це модель паралелізму по керуванню та модель загальної пам'яті.

У моделі загальної пам'яті паралельна програма являє собою систему потоків, що взаємодіють за допомогою загальних змінних та примітивів синхронізації [1].

Основна ідея моделі паралелізму по керуванню міститься в наступному. Замість програмування в термінах потоків пропонується розширити мови спеціальними керуючими конструкціями – паралельними циклами та паралельними секціями. Створення і знищення потоків, розподіл між ними витків паралельних циклів чи паралельних циклів (наприклад виклик процедур) – усе це бере на себе компілятор [1].

Метою роботи є розгляд та порівняння ефективності програм, що використовують різні моделі паралелізму.

Основна частина

У якості моделі загальної пам'яті була обрана модель Windows Threads, а також технологія Intel Thread Building Blocks, в якості моделі паралелізму по керуванню — технологія OpenMP. У якості тестового прикладу був обрано

алгоритм підрахунку числа π, що був реалізований на мові C++. Реалізація проводилась у середовище Visual Studio 2010.

Для розрахунку числа π, був обраний наступний алгоритм:

$$\pi = \int_0^1 \frac{4}{1+x}\,\mathrm{d}x \tag{1}$$

Для розрахунку визначеного інтегралу (1) використовувався метод трапецій, за яким:

$$\pi = \frac{1}{n}\,\Sigma_0^n \frac{4}{1+(\frac{2i+1}{2n})^2} \tag{2}$$

В усіх розрахунках за n приймалось число 0x7FFFFFFF = 2 147 483 647.

Розглянемо функції, за якими проводились розрахунки числа за формулою (2).

Послідовна функція:

```
double funcpi_1core(long accuracy)
{
    double h = 1. / accuracy;
    double res = 0;
    double x = 0;
    for(long i = 0; i < accuracy; i++)
    {
        x = (2 * i + 1) * h / 2;
        res += 4. / (1 + x * x);
    }
    res *= h;
    return res;
}
```

Використання технології Open MP. Так як тіло циклу *for* у функції *funcpi_1core* містить незалежні оператори, їх можна виконувати паралельно. Змінна *res* є загальною для усіх ітерацій циклу, значення цієї змінної накопичується (знак +), що реалізується за допомогою параметру *reduction* [2]. Змінена функція виглядає наступним чином:

```
double funcpi_omp(long accuracy)
{
    double h = 1. / accuracy;
    double res = 0;
    double x = 0;
#pragma omp parallel for reduction(+:res)
```

```
    for(long i = 0; i < accuracy; i++)
    {
        x = (2 * i + 1) * h / 2;
        res += 4. / (1 + x * x);
    }
    res *= h;
    return res;
}
```

Використання потоків Windows. Для паралельного виконання необхідно визначити потокову функцію. Потокова функція має виконувати порцію ітерацій циклу. Так як потокову функцію викликає операційна система, її заголовок фіксований і має наступний вигляд:

```
DWORD WINAPI Fun (PVOID par);
```

Таким чином, потоковій функції переда'ється один параметр, який є адресом даного довільного типу.

Для створення потоку використовується функція:

```
HANDLE WINAPI CreateThread(
    __in_opt LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in SIZE_T dwStackSize,
    __in LPTHREAD_START_ROUTINE lpStartAddress,
    __in_opt LPVOID lpParameter,
    __in DWORD dwCreationFlags,
    __out_opt LPDWORD lpThreadId
);
```

Для виконання ітерації циклу в потоковій функції їй необхідно передати: початкове та кінцеве значення параметру циклу, значення для підрахунку шагу інтегрування та адресу для результату підрахунку потокової функції [3]. Для передачі параметрів була створена відповідна структура:

```
struct PI_STRUCT
{
    long start;
    long end;
    long accuracy;
    double *res;
};
```

У цій структурі *start*, *end* – початкова і кінцева межі діапазону, у якому проводиться розрахунок значення, *accuracy* – значення для розрахунку кроку інтегрування, *res* – вказівник на область пам'яті, у яку заноситься результат.

Область пам'яті, у яку заноситься результат, є загальною для усіх потоків. Для синхронізації доступу до неї, використовувалась критична секція (CRITICAL_SECTION), це рішення засновувалось на тому, що критична секція на відміну від Mutex та Event, не є об'єктом ядра, й, таким чином, доступ до неї вимагає менше часу [4]. Повний лістинг функції виглядає наступним чином:

```
void pi_winthread(void* range)
{
    PI_STRUCT* rang = (PI_STRUCT*) range;
    double h = 1. / rang->accuracy;
    double res = 0;
    double x = 0;
    for (long i = rang->start; i < rang->end; i++)
    {
        x = (2 * i + 1) * h / 2;
        res += 4. / (1 + x * x);
    }
    res *= h;
    EnterCriticalSection(synh);
    *rang->res += res;
    LeaveCriticalSection(synh);
}
```

При виклику функції, необхідно враховувати, що створення потоків, передача значень повністю залежать від програміста. Виклик цієї функції у декількох потоках виглядає наступним чином:

```
synh = new CRITICAL_SECTION;
InitializeCriticalSection(synh);
HANDLE* h = new HANDLE[Cores];
PI_STRUCT* pistruct = new PI_STRUCT[Cores];
double* res = new double;
*res = 0;
int a = ACCURACY / Cores;
for(int i = 0; i < Cores; i++)
{
    pistruct[i].start = i * a;
    pistruct[i].end = (i + 1) * a;
    pistruct[i].accuracy = ACCURACY;
    pistruct[i].res = res;
```

```
        h[i] = (HANDLE)_beginthread(pi_winthread, 0, (void *) pistruct[i]);
    }
    WaitForMultipleObjects(Cores, h, true, INFINITE);
```

де *Cores* — кількість ядер, *ACCURACY* — кількість ітерацій.

При створенні функції, що використовує бібліотеку Intel Thread Building Blocks (TBB), була використана бібліотека від 15.03.2011. Написання функції з використанням TBB має ряд особливостей.

Концепція бібліотеки TBB міститься в тому, що розробнику необхідно створювати задачі, що виконують певні дії, які можуть бути розпаралелені. Кожна задача виноситься до окремого класу, звертання до задачі відбувається через перевантажений оператор "()".

У розпаралелену ділянку коду входить змінна, до якої мають загальний доступ на запис декілька процесів, щоб попередити ефект "гонок", ми використовуємо цикл з накопиченням — *parallel_reduce* [5].

Реалізована програма виглядає наступним чином:

```
#include "tbb.h"
#include "tbb_reduce.h"
#include "tbb_range.h"
using namespace tbb;
double res = 0;
double h = 0;
class ClassPi
{
    long accuracy;
    double res;
public:
    ClassPi (const long _accuracy): accuracy(_accuracy), res(0)
    ClassPi (const ClassPi c, split): accuracy(c.accuracy), res(0);
    void operator () (const blocked_range <long> r)
    {
        double h = 1. / accuracy;
        for(int i = r.begin(); i != r.end(); i++)
        {
            double x = (2 * i + 1) * h / 2;
            res += 4. / (1 + x * x);
        }
        res *= h;
    }
```

```
        void join (const ClassPi c)
        {
            res += c.res;
        }
        float gets ()return res;
};
int _tmain(int argc, _TCHAR* argv[])
{
        long accuracy = 0x7fffffff;
        task_scheduler_init init(8);
        ClassPi s = ClassPi(accuracy);
        parallel_reduce(blocked_range<long>(0, accuracy), s);
        init.terminate();
        return s.gets();
}
```

Кожна функція була скомпільована для виконання у 1, 2, 4 та 8 потоках в окрему програму. Кожна програма, для зменшення похибок, була запущена 10 раз на комп'ютері з процесором Intel Core i5 2,55MHz@4. Для підрахунку часу, що було затрачено на виконання кожної функції, використовувався зовнішній профіліровщік, що був написаний А. Станкевичем(С-Петербург). Програми викликалися за допомогою .bat скрипта. Результати виконання заносились до файлу-логу. Отримані результати можна побачити у таблиці 1. Усередненні отримані результати можна побачити на рисунку 1.

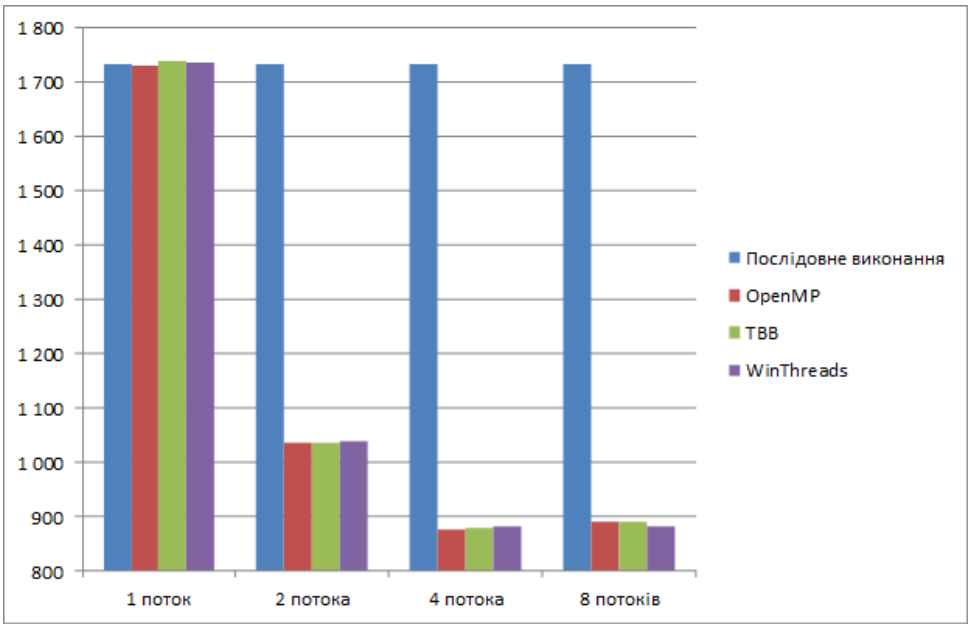

Рис. 1: Усередненні результати виконання програми.

	Спроби (мс)										
	1	2	3	4	5	6	7	8	9	10	Середнє
Послідовне	1739	1748	1728	1732	1734	1736	1728	1728	1725	1741	1734
omp 1 потік	1739	1729	1724	1725	1723	1726	1737	1732	1731	1732	1730
omp 2 потоки	1039	1040	1035	1035	1027	1044	1033	1024	1032	1041	1035
omp 4 потоки	870	873	875	880	880	885	873	870	873	871	875
omp 8 потоків	862	862	939	929	863	864	870	932	894	872	889
tbb 1 потік	1736	1738	1740	1740	1742	1735	1743	1739	1735	1732	1738
tbb 2 потоки	1021	1055	1045	1040	1037	1044	1040	1030	1030	1028	1037
tbb 4 потоки	877	877	874	871	881	887	879	880	879	881	879
tbb 8 потоків	932	937	871	892	872	875	885	881	883	879	891
win 1 потік	1736	1745	1734	1729	1740	1742	1737	1734	1735	1734	1737
win 2 потоки	1030	1030	1046	1038	1035	1036	1059	1036	1033	1033	1038
win 4 потоки	887	882	884	881	876	881	883	889	881	875	882
win 8 потоків	888	885	880	882	888	889	884	877	875	880	883

Табл. 1: Результати виконання скрипта.

Висновки

Кожна технологія має свій ряд особливостей. WinThreads є стандартною технологією в створенні паралельних за стосунків для ОС Windows, проте програмісту, при написанні коду, потрібно самому відповідати за створення потоків та їх синхронізацію, за швидкістю ця технологія поступається конкурентам, також не варто забувати, що дане рішення є платформо залежним.

TBB є доволі потужним інструментом, для паралелізації за стосунків та містить платформо незалежні шаблонні рішення для створення распаралелених циклів, об'єктів синхронізації, проте використання цієї технології можливо лише при написанні коду програми на мові C++. Також використання за стосунків, що були написані за допомогою TBB, вимагає наявності бібліотеки tbb.dll. По швидкості роботи код написаний за допомогою TBB, поступається OpenMP, проте виконується швидше за рішення WinThreads, проте написа-

ння цього коду вимагає від програміста знання особливостей використання бібліотеки TBB.

OpenMP являє собою стандарт, що описує сукупність директив компілятору, бібліотечних процедур та змінних оточення, які дозволяють розпаралелити будь-який послідовний алгоритм, що містить незалежні цикли та участки коду написаний на мові C, C++, Fortran майже без зміни коду, при цьому розпаралелений алгоритм буде мати оптимальний час виконання. Проте цей стандарт має обмежену функціональність, що не дає можливості розпаралелювати цикли типу, відмінного від for, створювати конвеєр та ін.

Література

[1] *Эндрюс Г.Р.* Основы многопоточного, параллельного и распределённного программирования / Г.Р.Эндрюс. – М., СПб. Киев: Вильямс, 2003. – 506с.

[2] *Chapman B.* Using OpenMP: portable shared memory parallel programming / Chapman B., Jost G., van Deras R. — Cambridge, Massachusets, London: The MIT Press, 2008. — 354 p.

[3] CreateThread Function (Windows) [Електроний ресурс]. — Режим доступу: http://msdn.microsoft.com/en-us/library/ms682453(v=vs.85).aspx

[4] Critical Section Objects (Windows) [Електроний ресурс]. — Режим доступу: http://msdn.microsoft.com/en-us/library/ms682530(VS.85).aspx

[5] *Reinders J.* Intel Threading Building Blocks / Reinders J. — Beijing, Cambridge, Farnham, Köln, Paris, Sebastopol, Taipei, Tokyo: O'Reilly, 2007. — 306p.