Технічні науки

УДК 004.432.2

Цой Ігор Сергійович

студент

Харківського національного університету радіоелектроніки

Цой Игорь Сергеевич

студент

Харьковского национального университета радиоэлектроники

Tsoi Ihor

Student of the

Kharkiv National University of Radioelectronics

Олійник Олена Володимирівна

старший викладач кафедри програмної інженерії

Харківський національний університет радіоелектроніки

Олейник Елена Владимировна

старший преподаватель кафедры программной инженерии

Харьковский национальный университет радиоэлектроники

Oliinik Olena

Assistant Professor of Software Engineering Department

Kharkiv National University of Radioelectronics

ВИКОРИСТАННЯ ПАРАЛЕЛІЗМУ У ПЛАТФОРМІ .NET CORE 3.1

ИСПОЛЬЗОВАНИЕ ПАРАЛЛЕЛИЗМА В ПЛАТФОРМЕ .NET CORE 3.1

PARALLEL PROGRAMMING IN .NET CORE 3.1

Анотація. *Досліджено можливості використання платформи .NET Core 3.1 для створення програмних додатків з використанням паралелізму.*

Ключові слова: *паралельні обчислення, цикли, потоки, SIMD команди.*

Аннотация. *Исследованы возможности использования платформы .NET Core 3.1 для создания программных приложений с использованием параллелизма.*

Ключевые слова: *параллельные вычисления, циклы, потоки, SIMD команды.*

Summary. *Investigation of the possibilities of parallel programming in .NET Core 3.1.*

Key words: *parallel computations, loops, threads, SIMD commands.*

.NET Core – це модульна платформа для розробки програмного забезпечення з відкритим вихідним кодом. Платформа була розроблена компанією Microsoft. Додатки розроблені для .NET Core можуть бути виконані на наступних операційних системах: Windows, Linux та macOS.

Зараз платформа .NET має велику популярність у сфері серверної розробки, розробки настільних додатків для ОС Windows, розробки кросплатформених мобільних додатків та у сфері розробки хмарних сервісів. Платформа дозволяє створювати програмні додатки з використанням різних видів паралелізму, надає конструкції для створення потоків, їх синхронізації та їх конфігурування.

Додатки для платформи .NET Core можуть бути розроблені з використання множини .NET-сумісних мов. Найпопулярнішою мовою для створення програмних додатків на платформі .NET Core є мова C#.

Одним з видів паралелізму є використання SIMD команд [1]. SIMD команди реалізовані на рівні процесору. Вони дозволяють одночасно обробляти масив даних певного типу певною командою. Наприклад, можна отримати квадратний корінь для кожного елементу масиву чисел за одну операцію.

Для використання SIMD операцій в .NET Core існує тип Vector<T>. У якості параметру типу T можуть бути використані будь-які числові типи (short, int, long тощо).

Щоб дізнатися над скількома елементами певного типу можуть бути одночасно виконані SIMD операції використовується статична властивість класу Vector<T> – Count.

Завантажити значення у вектор можна за допомогою конструктору, який приймає у якості параметрів масив значень та індекс першого значення для завантаження. Якщо бажано отримати вектор з усіма елементами рівними 0, можна використати статичну властивість класу Vector<int> – Zero. Якщо всі елементи повинні дорівнювати одиниці – One.

Для прикладу застосування SIMD команд та визначення прискорення роботи додатків за рахунок використанням SIMD команд порівнюється операція додавання елементів масиву лінійним алгоритмом та алгоритмом з використанням SIMD команд. Код функції додавання елементів масиву лінійним алгоритмом:

```
static int SumNaive(int[] integers)
{
	int sum = 0;
	for (int i = 0; i < integers.Length; i++)
		sum += integers[i];

	return sum;
}
```

Код функції додавання елементів масиву алгоритмом з використанням векторних операцій:

```
static int SumVector(int[] integers)
{
	int vectorSize = Vector<int>.Count;
	var accVector = Vector<int>.Zero;
	int i;
	var array = integers;
	for (i = 0; i < array.Length - vectorSize; i += vectorSize)
```

```
	{
		var v = new Vector<int>(array, i);
		accVector = Vector.Add(accVector, v);
	}
	int result = Vector.Dot(accVector, Vector<int>.One);
	for (; i < array.Length; i++)
		result += array[i];

	return result;
}
```

Результат експерименту: використання SIMD операцій дає прискорення в 4 рази при розмірі вхідного масиву – 100'000'000 елементів (82 мс – лінійний алгоритм, 20 мс – SIMD алгоритм). Проте при розмірі масиву 1'000'000 – прискорення не маємо. При розмірі 100'000 елементів час додавання елементів масиву лінійним алгоритмом є меншим (0,4 мс – лінійний алгоритм, 1 мс – SIMD алгоритм). Це пов'язано з накладними витратами часу на створення векторів, копіювання елементів в вектор що зберігає проміжні результати. Ці витрати часу нівелюються при великому розмірі вхідного масиву, але при малому розмірі – лише знижують показники часу.

Іншим методом оптимізації програмних додатків є паралелізація циклів за рахунок використання потоків. Якщо додаток виконується на багатоядерній системі, потоки можуть виконуватися паралельно. Для паралельного виконання циклів існує статичний метод For класу Parallel [2]. У якості параметрів цей метод отримує початковий номер ітерації, наступний після останнього номер ітерації та функцію, яка повинна бути виконана для кожної ітерації. Ітерації циклу рівномірно розподіляються між потоками, кількість яких дорівнює кількості логічних ядер процесора. Таким чином, програміст може створювати паралельний код не працюючи безпосередньо з потоками. У якості прикладу використання методу For класу Parallel розглядається задача підрахунку кількості у масиві степенів трійки.

Функція-предикат:

```
static bool IsPowerOfThree(int integer)
{
	if (integer == 0)
		return false;
	while (integer % 3 == 0)
		integer /= 3;

	return integer == 1;
}
```

Код функції підрахунку степенів трійки у масиві лінійним алгоритмом:

```
static int CountPowersOfThreeSequentially(int[] integers)
{
    int count = 0;
    for (int i = 0; i < integers.Length; i++)
    {
      if (IsPowerOfThree(integers[i]))
		count++;
    }
    return count;
}
```

Код функції підрахунку степенів трійки у масиві з використанням Parallel.For:

```
static int CountPowersOfThreeInParallel(int[] integers)
{
	int count = 0;
	Parallel.For(0, integers.Length,
		localInit: () => 0,
		body: (int index, ParallelLoopState _, int localSum) =>
			IsPowerOfThree(integers[index]) ? (localSum + 1) :
			localSum,
		localFinally: (int finalLocalSum) =>
			Interlocked.Add(ref count, finalLocalSum));
	return count;
}
```

При невеликих розмірах масиву однопоточне рішення працює швидше, бо накладні витрати на паралелізацію суттєво перевищують час

виконання кожної ітерації. Але при великих розмірах масиву, за рахунок паралельного виконання ітерацій, багатопоточне рішення працює значно швидше – з'являється суттєве прискорення (час зменшується у 3,5 рази).

Іншим способом оптимізації програм за рахунок паралелізму є виконання секцій коду у різних потоках. Клас Parallel має статичний метод Invoke [3]. Цей метод дозволяє паралельно запустити декілька функцій та дочекатися завершення їх виконання. Функції будуть рівномірно розподілені між потоками, кількість яких дорівнює кількості логічних ядер процесора.

Використання паралелізації секцій коду є дуже зручним методом оптимізації якщо існує декілька різних задач які можуть виконуватися незалежно. При наявності однакових задач, більш зручним та логічним рішенням є реорганізація коду, після якої можна буде застосувати паралелізацію циклів.

Наразі, програмні додатки розроблені на платформі .NET Core не можуть досягати таких самих показників ефективності, як нативні додатки розроблені на мовах C чи C++. Це пов'язано з тим як влаштована платформа, а саме з залежністю від середовища виконання, наявністю збірки сміття тощо. Проте рівень складності розуміння, підтримки та тестування коду для платформи .NET Core є значно нижчим порівняно з мовами C чи C++. Ця властивість дозволяє більш швидко розробляти та розвивати програмні додатки.

Платформа гарно показує себе у сфері серверних додатків, що безумовно каже про високу якість реалізації у ній можливостей паралельного програмування. Отже платформа .NET Core може бути успішно використана для додатків які потребують паралельного виконання.

Література

1. Качко Е. Г. Параллельное программирование: Учебное пособие / Е. Г. Качко. Харьков: Форт, 2011. 528 с.
2. Parallel Programming in .NET. URL: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming
3. Parallel.Invoke method documentation. URL: https://docs.microsoft.com /en-us/dotnet/api/system.threading.tasks.parallel.invoke