Технічні науки

УДК 004.051

Середа Дар'я Антонівна

студентка

Харківського національного університету радіоелектроніки

Середа Дарья Антоновна

студентка

Харьковского национального университета радиоэлектроники

Sereda Daria

Student of the

Kharkiv National University of Radio Electronics

Квасняк Катерина Миколаївна

студентка

Харківського національного університету радіоелектроніки

Квасняк Екатерина Николаевна

студентка

Харьковского национального университета радиоэлектроники

Kvasniak Kateryna

Student of the

Kharkiv National University of Radio Electronics

Науковий керівник:

Олійник Олена Володимирівна

старший викладач кафедри ПІ

Харківський національний університет радіоелектроніки

РОЗПОДІЛ РОБОТИ ПІД ЧАС РОЗРОБКИ ПАРАЛЕЛЬНИХ ПРОГРАМ В OPENMP

РАСПРЕДЕЛЕНИЕ РАБОТЫ ВО ВРЕМЯ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ В OPENMP
DISTRIBUTION OF WORK DURING THE DEVELOPING OF PARALLEL PROGRAMS IN OPENMP

Анотація. *Досліджено теоретичні питання щодо розподілу роботи між потоками в OpenMP.*

Ключові слова: *розпаралелювання роботи потоків, синхронізація потоків, функції розподілу роботи.*

Аннотация. *Исследованы теоретические вопросы по распределению работы между потоками в OpenMP.*

Ключевые слова: *распараллеливание работы потоков, синхронизация потоков, функции распределения работы.*

Summary. *The theoretical aspects of distribution between threads in OpenMP were investigated.*

Key words: *parallelization of work of threads, synchronization of threads, functions of work distribution.*

У сфері програмування існує безліч засобів розпаралелювання програм - від функцій операційної системи до спеціальних бібліотек, які розроблені для реалізації паралельних процесів та взаємодіючих потоків. Для створення багатопоточних програм на системах, що мають спільну пам'ять, використовується специфікація OpenMP - стандарт, який визначає набір директив компілятора та змінних середовища [1, с. 6-9].

Для того, що дозволити виконання коду в паралельному режимі, розробникові застосунку необхідно явно вказати директиви OpenMP. Взагалі, можна виділити 3 типи директив: директиви синхронізації, паралельної області та розподілу роботи. Кожна з них може мати окремі

опції, які ще називають атрибутами директиви. Їх кількість та перелік залежить від специфікації самої програми. Порядок опцій не важливий, адже у рамках однієї директиви опція може повторюватися. Також слід звернути увагу на те, що після деяких опцій може слідувати список змінних, розділених комами.

Розподіляти роботу в OpenMP можна по-різному, навіть на найнижчому рівні. Провівши не одне дослідження щодо можливостей використання різних методів розподілу роботи в паралельних програмах, ми з'ясували певні особливості директиви та її основні функції. Одними з найважливіших виявилися *omp_get_num_threads()* *ma omp_get_thread_num()*. Виклик функції *omp_get_thread_num()* повертає унікальний номер потоку у паралельній області. Функція *omp_get_num_threads()* повертає кількість усіх потоків, які одночасно виконують фрагмент коду. Їх використання та процес розподілу роботи в програмі з паралельним виконанням краще розглянути на прикладі нашого дослідження. Розглянемо використання функцій omp_get_num_threads() та omp_get_thread_num() на рисунку 1.

```
5   #include <iostream>
6   #include <omp.h>
7
8   int main()
9   {
10      int countOfThreads, threadNum;
11  #pragma omp parallel
12      {
13          countOfThreads = omp_get_num_threads();
14          threadNum = omp_get_thread_num();
15          if (threadNum == 0) {
16              printf("Кількість потоків %d\n", countOfThreads);
17          }
18          else {
19              printf("Поток %d\n", threadNum);
20          }
21      }
22  }
```

Рис. 1. Частина коду, яка демонструє використання функцій omp_get_num_threads() та omp_get_thread_num() в директиві parallel

Код програми написаний мовою С. У нашому експерименті щодо використання функцій *omp_get_num_threads() та omp_get_thread_num()* в ОС Windows 10 та процесором з 8-ми логічними ядрами (потоками) отримуємо, що кожен потік має свою частину коду для виконання і саме таким чином між потоками розподіляється робота [2, с. 40-48]. Але вони не забезпечують автоматичну організацію синхронізації доступу до загальних даних.

Якщо змінна середовища OMP_DYNAMIC дорівнює true, то система може динамічно змінювати кількість потоків, що будуть використовуватись для виконання паралельної області, наприклад, для покращення оптимізації використання ресурсів системи.

Дізнатися значення змінної середовища OMP_DYNAMIC можливо за допомогою функції виклику omp_get_dynamic().

Мовою С параметром функції omp_set_dynamic() вказується 0 або 1. Якщо ж система не підтримує динамічну зміну кількості потоків, то при виклику функції omp_set_dynamic() значення змінної OMP_DYNAMIC не зміниться [3, с. 35-37].

Шляхом експериментів нам вдалось виявнити, що паралельні області бувають вкладеними. По дефолту вкладена паралельна область виконується одним потоком. Це керується встановленням змінної середовища OMP_NESTED. Змінити значення змінної OMP_NESTED можливо за допомогою функції omp_set_nested(). Ця функція дозволяє або забороняє вкладений паралелізм. Якщо вкладений паралелізм дозволений, кожний потік, де трапиться опис паралельної області, створить нову групу потоків для його виконання. Потік, який створив групу, при цьому стане в ній потоком-майстром.

```
    int numThread;
    omp_set_nested(1);
#pragma omp parallel private(n)
    {
        numThread = omp_get_thread_num();
#pragma omp parallel
        {
            printf("Приклад 1, потік %d - %d\n", numThread, omp_get_thread_num());
        }
    }
    omp_set_nested(0);
#pragma omp parallel private(n)
    {
        numThread = omp_get_thread_num();
#pragma omp parallel
        {
            printf("Приклад 2, потік %d - %d\n", numThread, omp_get_thread_num());
        }

    }

}
```

Рис. 2. Код, який показує використання функції omp_set_nested()

Даний код програми демонструє використання вкладених паралельних областей і функції omp_set_nested(). Перед першим прикладом виклик функції omp_set_nested() вирішує використання вкладених паралельних областей. Щоб визначити номер потоку в паралельній секції, використовуємо вже знайому нам функцію omp_get_thread_num(). В результаті потік зовнішньої паралельної області створює нові потоки, кожний з яких виводить свій номер разом з номером потоку, який його створив. Далі ф-ція omp_set_nested() забороняє використання вкладених паралельних областей. У другому ж прикладі вкладена паралельна область виконуватиметься без породження нових потоків.

Отже, провівши дослідження, ми вияснили, що розподіл роботи між запущеними потоками в OpenMP відбувається за допомогою багатьох функцій. Ключові елементи набору директив компілятора допомагають керувати не тільки потоками, але і даними, часом виконання програми та поведінкою паралельної програми в цілому.

Література

1. Антонов А.С. Параллельное программирование с использованием технологии OpenMP // Изд-во МГУ. 2009. С. 77.
2. Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ // ДМК Пресс. 2012. С. 672.
3. Корнеев В.В. Параллельные вычислительные системы // Изд-во "Нолидж". 1999.