Vol. 5, No. 1, 2023

Андрій Андрушко[1], Олександр Маркелов[2]

[1]Кафедра систем автоматизованого проектування, Національний університет "Львівська політехніка", вул. Степана Бандери 12, Львів, Україна, E-mail: andrii.m.andrushko@lpnu.ua, ORCID 0000-0003-4229-7589
[2]Кафедра систем автоматизованого проектування, Національний університет "Львівська політехніка", вул. Степана Бандери 12, Львів, Україна, E-mail: oleksandr.e.markelov@lpnu.ua, ORCID 0000-0002-2432-0768

ОРГАНІЗАЦІЯ БАГАТОПОТОКОВИХ ОБЧИСЛЕНЬ В C++

Отримано: липень 17, 2023 / Переглянуто: вересень 28, 2023 / Прийнято: жовтень 10, 2023

https://doi.org/

Анотація. Протягом багатьох років, збільшення обчислювальної потужності сучасних пристроїв досягається не через підвищення тактової частоти та пропускної здатності процесорів, а шляхом застосування гіперпотокових і багатоядерних архітектур. Ця проста зміна підходу до дизайну комплектуючих призвела до драматичних змін в організації обчислень і стала поворотним пунктом для розробників програмного забезпечення. Програмне забезпечення, яке має скористатися збільшенням обчислювальної потужності багатоядерних архітектур, повинно бути розроблена таким чином, щоб мати змогу одночасно виконувати кілька завдань. При висвітленні теми паралельних/одночасних обчислень науковці та фахівці в галузі ІТ користуються двома термінами: (1) «паралелізм» (англ. parallelism) та (2) «одночасне/узгоджене виконання» (англ. concurrency). Третім важливим терміном при розгляді паралельних/одночасних обчислень, є «багатопотоковість» (multithreading).

У C++ двома найпоширенішими способами реалізації паралелізму є узгоджене виконання та власне паралелізм. Хоча їх можна використовувати і в інших мовах програмування, C++ виділяється своїми можливостями використання одночасних обчислень з нижчим, ніж середнє, залученням загальних ресурсів машини, а також здатністю виконувати складні інструкції. У стандарті C++11 була запропонована підтримка багатопотокових програм. Стандарт C++ визнавав існування багатопотоковості у мові та надавав компоненти для написання багатопотокових програм у бібліотеці <thread>. Це зробило можливим написання багатопотокових програм на C++, не покладаючись на специфічні для певної платформи розширення, та дало змогу створювати портативний багатопотоковий код із гарантованою поведінкою. Бібліотека <thread> надає широкі можливості для організації паралельних та узгоджених/одночасних обчислень, містить також модель пам'яті C++, умовні змінні, м'ютекси та ін. для синхронізації роботи потоків. У статті здійснено аналіз можливостей написання програм мовою C++ з використанням кількох потоків для паралельного та/або узгодженого виконання завдань, а також розгляд функцій мови C++ і засобів бібліотеки <thread>, які роблять це можливим.

Ключові слова: C++, багатопотоковість, паралелізм, одночасне/узгоджене виконання, синхронізація потоків

Вступ та постановка задачі

Ще з початку 2000-х років, основні виробники процесорів і програмних архітектур, від Intel і AMD до Sparc і PowerPC, вичерпали більшість своїх традиційних підходів до підвищення продуктивності центрального процесора (CPU) [1]. Замість того, щоб підвищувати тактову частоту та пропускну здатність для обробки послідовних прямолінійних інструкцій, вони натомість масово звертаються до гіперпотокових і багатоядерних архітектур. Обидві ці функції вже доволі тривалий

час є реалізованими у відповідних комплектуючих та мікросхемах – багатоядерні комп'ютери та ноутбуки, і навіть багатоядерні гаджети завоювали вже багато ринків і стали невід'ємною нормою життя. Збільшення обчислювальної потужності сучасних пристроїв має місце не через виконання одного завдання швидше, а через виконання кількох завдань паралельно. Паралельні обчислення, колись лише ніша для вчених та інженерів-науковців, тепер стали повсякденною реальністю [2].

Зміна підходу до дизайну комплектуючих призвела до драматичних змін в організації обчислень і стала поворотним пунктом для розробників програмного забезпечення. Якщо раніше можна було написати програму і сподіватись, що вона працюватиме швидше із виходом нової, продуктивнішої моделі процесора, то тепер, за висловом Херба Саттера, «безкоштовний обід закінчився» [1]. Оскільки багатоядерні процесори стали домінуючими при виробництві комп'ютерних систем, використання багатопотоковості стає все більш важливим чинником прискорення роботи програм [3]. Багатопотокова технологія дозволяє програмам виконувати різні завдання одночасно, таким чином забезпечуючи повне використання обчислювальних ресурсів [4].

Програмне забезпечення, яке має скористатися цим збільшенням обчислювальної потужності новітніх процесорів, повинно бути розроблена таким чином, щоб мати змогу одночасно виконувати кілька завдань. Тому виробники програмних продуктів повинні чітко усвідомити цей факт і ті, хто донедавна нехтував використанням паралельних алгоритмів у своїх програмах, тепер повинні додати ці можливості до своїх умінь та компетентностей. Оскільки кількість ядер у процесорі комп'ютера збільшується з кожним роком, паралелізм безсумнівно залишиться безцінним активом в арсеналі сучасного розробника [5], а програмісти повинні мати навички в цьому контексті, оскільки вони стають необхідними для вивчення й застосування на практиці повного потенціалу нових комп'ютерних систем [6, ст. 23].

Розробники повинні використовувати багатопотоковість з кількох причин:

- Вища пропускна здатність;
- Програми із кращою швидкодією;
- Ефективніше використання ресурсів.

Зі стрімким розвитком багатоядерних машин інженери, які вміють орієнтуватися в їх складності, є сьогодні найбільш бажаними кандидатами для більшості технологічних компаній. Розуміння того, як працюють потоки, і знання принципів паралельного програмування демонструють зрілість і технічну глибину розробника [7].

У C++ двома найпоширенішими способами реалізації паралелізму є узгоджене виконання та власне паралелізм. Хоча їх можна використовувати і в інших мовах програмування, C++ виділяється своїми можливостями використання одночасних обчислень з нижчим, ніж середнє, залученням загальних ресурсів машини, а також здатністю виконувати складні інструкції [5].

Об'єктом дослідження є сутність принципи організації паралельних та одночасних/узгоджених обчислень у сучасній комп'ютерній науці.

Предметом дослідження є власне можливості реалізації паралельних та одночасних/узгоджених обчислень мовою програмування C++.

Метою даної статті є розгляд та аналіз функціоналу мови C++ і засобів бібліотеки <thread>, які роблять можливим написання програм із використанням кількох потоків для паралельного та/або узгодженого виконання завдань. Однак, перед тим як перейти до детального розгляду цього питання, необхідно також розглянути деякі аспекти термінології паралельних/багатопотокових обчислень, для того щоб уникнути плутанини при висвітленні матеріалу.

Результати та обговорення

Зазвичай при висвітленні теми паралельних/одночасних обчислень науковці та фахівці в галузі IT користуються двома термінами: (1) «паралелізм» (англ. parallelism) та (2) «одночасне/узгоджене виконання» (англ. concurrency) [8, 2, 7]. Варто зауважити при цьому, що доволі часто ці терміни використовуються як тотожні, незважаючи на те, що між ними існує суттєва відмінність.

Третім важливим терміном, який зазвичай використовується при розгляді

паралельних/одночасних обчислень, є «багатопотоковість» (multithreading) – «якщо ви хочете, щоб ваша програма повністю використовувала потенціал нових процесорів, вона має використовувати багатопотоковість» [1]. Спробуємо детальніше зупинитись на цих термінах та розібратись у чому ж їх сутність.

У найбільш простому розумінні, одночасне/узгоджене виконання стосується двох або більше видів діяльності, які відбуваються в один і той же час. Як вже було зазначено вище, одночасне/узгоджене виконання та паралелізм мають доволі подібне трактування коли ми говоримо про написання коду, який використовує багато потоків. В літературі зазвичай виділяють дві головні причини їх використання. Перша – це логічно розділити потоки, які по суті є незалежними; друга – це підвищити ефективність та швидкість обчислень [1].

Власне у цих двох головних причинах і полягає відмінність між термінами «одночасне/узгоджене виконання» та «паралелізм» (рис. 1). Відмітимо, що деякі автори вже чітко розділяють ці поняття і означують їх доволі конкретно. «Різниця проявляється у нюансах, фокусі та цілях використання потоків. Обидва терміни стосуються організації одночасного виконання кількох завдань при доступних засобах обчислення (комплектуючих). Паралелізм однак є більш орієнтованим на швидкодію, коли мова йде про оптимальне використання наявних обчислювальних ресурсів з метою пришвидшення роботи програмного забезпечення. У той час коли про одночасне/узгоджене виконання говорять тоді, коли першочерговим завданням є розділення відповідальності або чутливість (швидкість реагування) стосовно результатів виконання обчислень» [8].

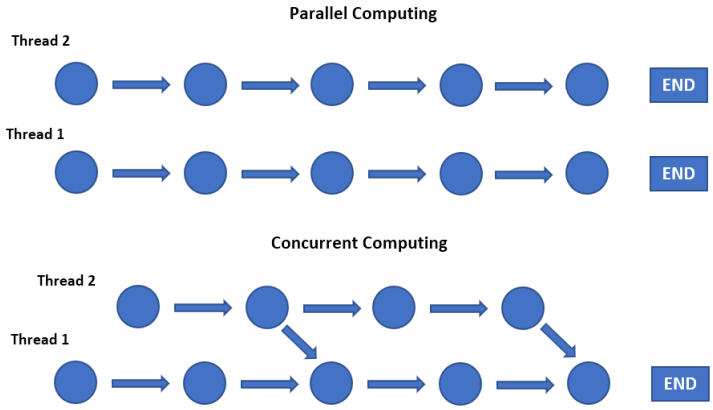

Рис. 1. Схема виконання одночасних/узгоджених та паралельних алгоритмів

Подібну думку висловлюють також Acar, Charguéraud та Rainey [2]: «Ми використовуємо термін «паралельний» для позначення алгоритму або програми, які виконують кілька обчислень одночасно з метою пришвидшення або зменшення часу роботи алгоритму/програми. Поведінку введення-виведення результатів або зовнішню семантику паралельного обчислення можна визначити як чисто послідовну, навіть при умові, що самі обчислення виконуються паралельно. Ми використовуємо термін «одночасний/узгоджений» для позначення обчислень, які включають незалежних агентів, що реалізуються за допомогою процесів або потоків, обмінюються даними та координуються для досягнення наміченого кінцевого результату. Поведінка введення-виведення результатів або зовнішня семантика одночасної/узгодженої програми не можуть бути визначені чисто послідовно; ми повинні враховувати взаємодію між різними потоками».

Одночасність/узгодженість – це здатність вашої програми мати справу (не виконувати) з багатьма речами одночасно і досягається за допомогою багатопотоковості. Не потрібно плутати одночасність/узгодженість із паралелізмом, який просто передбачає виконання багатьох речей одночасно [7].

Як вже було зазначено вище, для реалізації як одночасних/узгоджених обчислень, так і для реалізації паралельних обчислень використовується багатопотоковість – підхід до організації коду, який дозволяє одночасне (синхронне) виконання двох або більше частин програми для

максимального використання процесора або координації використання результатів обчислень. Багатопотоковість реалізується за допомогою процесів та/або потоків. Процеси це власне те, що виконує програма. Кожен процес може одночасно виконувати кілька підзавдань, які називаються потоками.

Термін «багатопотоковість» означає обчислення з кількома потоками керування. Після свого створення потік виконує обчислення доки не завершиться, дотримуючись послідовності інструкцій, визначених програмою [2]. Багатопотокове обчислення починається із запуску основного потоку, який є потоком, з якого починається виконання усієї програми (рис. 2). Потік може створювати або породжувати інший потік, синхронізуватися з іншими потоками за допомогою різноманітних конструкцій синхронізації, таких як блокування (locks), м'ютекс (mutex), змінні синхронізації та семафори (semaphores). Власне про специфіку реалізації багатопотоковості і піде мова у наступних параграфах статті.

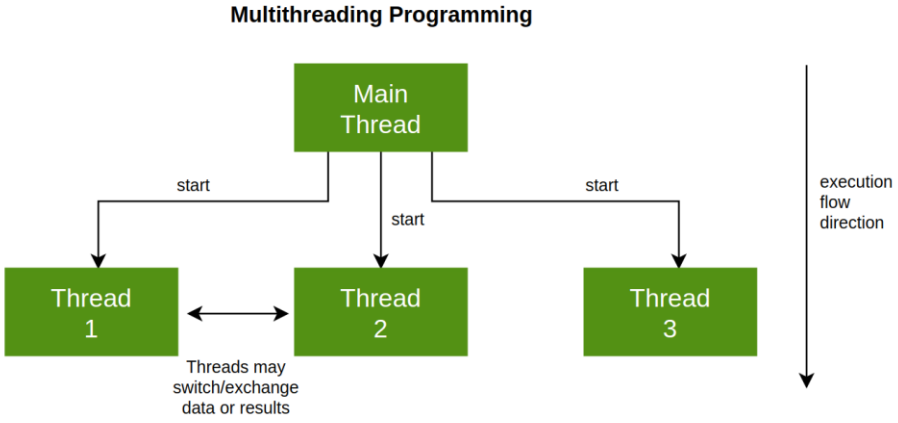

Рис. 2. Багатопотокова программа [9].

Бібліотека <thread> та організація потоків в C++.

1. Стандарт C++ 11 та підтримка багатопотокового програмування.

Тринадцять років після того, як було опубліковано перший стандарт мови C++, комітетом стандартів C++ було здійснено ґрунтовний перегляд багатьох засадничих принципів. Результатом такого перегляду став новий стандарт C++, опублікований у 2011 році [10]. У новому стандарті було запропоновано цілу низку змін, які зробили роботу в C++ легшою та продуктивнішою.

Однією з найважливіших нових функцій у стандарті C++11 була підтримка багатопотокових програм. Вперше за увесь час, стандарт C++ визнавав існування багатопотоковості у мові та надавав компоненти у бібліотеці для написання багатопотокових програм. Це зробило можливим написання багатопотокових програм на C++, не покладаючись на специфічні для певної платформи розширення, та дало змогу створювати портативний багатопотоковий код із гарантованою поведінкою [8]. «Підтримка багатопотоковості була представлена в стандарті C++11. До цього стандарту ми повинні були використовувати потоки POSIX або бібліотеку <pthreads>. Хоча ця бібліотека справлялася зі своєю роботою, відсутність будь-якого стандартного набору мовних функцій для організації потоків створювала серйозні проблеми для реалізації програмних продуктів на різних платформах. Стандарт C++ 11 покінчив із усім цим і дав нам std::thread»[11].

C++11 був першим стандартом C++, який представив можливості одночасної/узгодженої організації потоків, що містив також модель пам'яті C++, умовні змінні, м'ютекси та ін. [5]. Необхідно відмітити, що ця можливість з'явилась саме у той час, коли розробники, для підвищення продуктивності своїх програмних продуктів, почали все більше уваги звертати як на одночасність/узгодженість та паралелізм загалом, так і на багатопотокове програмування зокрема.

Наступні стандарти мови, C++14 і C++17, розширили цей базовий функціонал, щоб забезпечити подальшу підтримку написання багатопотокових програм на C++, а також запропонували опис технічних специфікацій цього функціоналу. Зокрема існують технічні

специфікації для розширень, що підтримують одночасність/узгодженість, а також для інших розширень, що підтримують паралелізм. Хоча останні були включені лише починаючи від стандарту С++ 17.

2. Бібліотека <thread>.

Бібліотека <thread> у С++ є частиною стандартної бібліотеки С++ і забезпечує функціональні можливості для створення та керування потоками в багатопотоковій програмі. Це дозволяє створювати кілька потоків виконання в одній програмі, кожен з яких виконується незалежно та одночасно. Щоб використовувати бібліотеку <thread>, вам потрібно включити заголовок <thread> у вашу програму С++: #include <thread>.

Розглянемо деякі ключові особливості та функції, які надає бібліотека <thread>:

1. *Створення потоку.* Бібліотека містить клас std::thread, за допомогою якого можна представити окремий потік виконання – «для незалежного виконання делегованих підзавдань, багатопотоковість С++ передбачає створення та використання об’єктів потоку, які у коді оголошуються за допомогою виразу std::thread» [5]. Далі цьому об’єкту можна передати у якості аргументів функцію або лямбда вираз.

```
voidmyFunction(param) {
   // визначення функції
}
std::threadmyThread(myFunction, params); // створення нового потоку, що використовує функцію
                                // параметри для функції вказуються після коми
std::threadmyThread([](){              // створення потоку, що використовує лямбда вираз
std::cout<< "thread function\n";
   });
```

2. *Керування потоками.* Бібліотека <thread> дозволяє вам керувати потоками, надаючи функції для приєднання та від’єднання потоків. Коли потік є приєднаним, головний потік (той що викликав приєднаний потік) очікує поки означений приєднаний потік завершить виконання. Коли потік від'єднується, він продовжує виконання незалежно від головного потоку.

```
myThread.join();    // приєднання потоку
myThread.detach();  // від’єднання потоку
```

3. *Синхронізація потоків.* Хоча багатопотоковість і створює багато зручностей, вона також може мати і деякі небажані наслідки. Наприклад, коли кілька потоків читають та записують ті самі дані одночасно. Один потік може ще не закінчити запис даних, коли іншому вже потрібно модифікувати або зчитувати дані – це може призвести до плутанини в доступі до даних. Бібліотека надає різні примітиви синхронізації, такі як м'ютекси, умовні змінні та змінні елементарного рівня, які дозволяють синхронізувати доступ до спільних ресурсів і координувати зв'язок між потоками.

```
#include <mutex>
#include <condition_variable>
#include <atomic>

std::mutex myMutex;  // створення м’ютексу
myMutex.lock();
  ……// сегмент коду для якого потрібна синхронізація
myMutex.unlock();

std::condition_variablemyCondition;  // створення умовної змінної
myCondition.wait(lock);
myCondition.notify_all();

std::atomic<int>myAtomic(0);  // створення змінної елементарного рівня
myAtomic.fetch_add(1);
```

4. *Ідентифікація потоку.* Бібліотека дає можливість отримати інформацію про поточний потік,

наприклад його унікальний ідентифікатор, за допомогою відповідних функцій.

```
std::thread::idthreadId = std::this_thread::get_id();
```

3. Реалізація програми з використанням бібліотеки <thread>.

Для ілюстрації роботи бібліотеки <thread> розглянемо програму представлену нижче.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <thread>
#include <mutex>
#include <condition_variable>

using namespace std;

vector <vector<double>>rows;
mutex mtx;                          // Mutex for synchronization
condition_variablecv;                // Condition variable for thread synchronization
bool finished = false;               // Flag to indicate the end of file

void readFile(ifstream& file);
void calculateSD();

int main() {

ifstream file("data.txt");
  if (!file.is_open()) {
cerr<< "Failed to open the file." <<std::endl;
    return 1;
  }

  thread readerThread(readFile, ref(file));
  thread calculatorThread(calculateSD);

readerThread.join();
calculatorThread.join();
  return 0;
}
void readFile(ifstream& file) {
  string line;
  while (getline(file, line)) {
    vector<double>row;
istringstreamiss(line);
    double value;
    while (iss>> value) {
row.push_back(value);
    }
rows.push_back(row);
cv.notify_one(); // Notify the second thread and continue reading the next row simultaneously
  }
  finished = true;
cv.notify_one();  // Notify the second thread that file reading is finished
}
void calculateSD() {
  while (true) {
```

```
unique_lock<mutex> lock(mtx);
cv.wait(lock, [] { return !rows.empty() || finished; }); // Wait until the first thread
                                                         // notifies
   if (finished &&rows.empty()) {
break;                        // All rows processed, exit the thread
   }
   vector<double> row = rows.back();
rows.pop_back();
   double sum = 0;                      // Calculate average for the row
   for (double value : row) {
     sum += value;
cout<< value << " ";
   }
   double average = static_cast<double>(sum) / row.size();
   double sumdev = 0;                        // Calculate standard deviation for the row
   for (double value : row) {
sumdev += ((value - average) * (value - average));
   }
   double stdev = sqrt(sumdev / (row.size() - 1));
cout<< "   Average for row: " << average << "    Standard deviation: " <<stdev<<endl;
  }
}
```

Програма обраховує масив дійсних додатних чисел за допомогою двох синхронізованих потоків. Перший потік зчитує масив із файлу “data.txt” і записує прочитані рядки у вектор. Другий потік розраховує середнє значення елементів та стандартне відхилення для кожного рядка. Потоки створюються і приєднуються у функції main(), а функції для читання і обчислення передаються як параметри.

```
thread readerThread(readFile, ref(file));
thread calculatorThread(calculateSD);
readerThread.join();
calculatorThread.join();
```

Реалізація потоків здійснюється узгоджено – як тільки перший потік (readerThread)прочитав рядок із файлу, він негайно повідомляє про це другому потоку (calculatorThread), з тим щоб другий потік починав обчислення. Повідомлення здійснюється за допомогою умовної змінної cv та функції notify_one() (cv.notify_one()).Після першого повідомлення потік продовжує далі читати рядки і надсилати повідомлення після кожного прочитаного рядка. Таким чином потоки працюють майже одночасно/паралельно, що у порівнянні із послідовним виконанням, дає змогу пришвидшити здійснення розрахунків.

Другий потік є замкнутим за допомогою mutex (unique_lock<mutex>lock(mtx)) поки не отримає повідомлення від першого потоку, що перший рядок прочитано і можна починати обчислення. Він очікує на повідомлення використовуючи функцію wait():

```
cv.wait(lock, [] { return !rows.empty() || finished; });
```

Отримавши повідомлення, другий потік починає обчислення першого рядка. Закінчивши обчислення першого рядка, потік знову замикається очікуючи на повідомлення від першого потоку про прочитання другого рядка. Цикл повторюється поки не буде прочитано усі дані з файлу. Після прочитання, змінна finished в першому потоці приймає значення true, і другий потік повідомляється про це за допомогою функції notify_one ( ).

Таке узгодження роботи потоків (другий потік не починає обчислень без повідомлення від першого потоку про прочитання рядка) дає можливість забезпечити коректне виконання програми. Узгодження потоків виключає ситуації, коли рядок масиву ще не прочитаний, або прочитаний не повністю. Перша ситуація могла б потенційно призвести до передчасного завершення програми

(коли другий потік не має що рахувати), або швидше за все до помилки і збою програми; у другому випадку (коли рядок прочитаний не повністю), ми б отримали некоректні результати обчислень.

Для перевірки роботи було використано масив розміром 10x10 елементів. Елементи масиву не змінювались, для того щоб перевірити правильність результатів обчислень. Програма запускалась декілька разів, і кожного разу результати обчислень були однаковими (рис. 3). Це свідчить про те, що робота потоків є дійсно узгодженою і забезпечує коректне виконання програми.

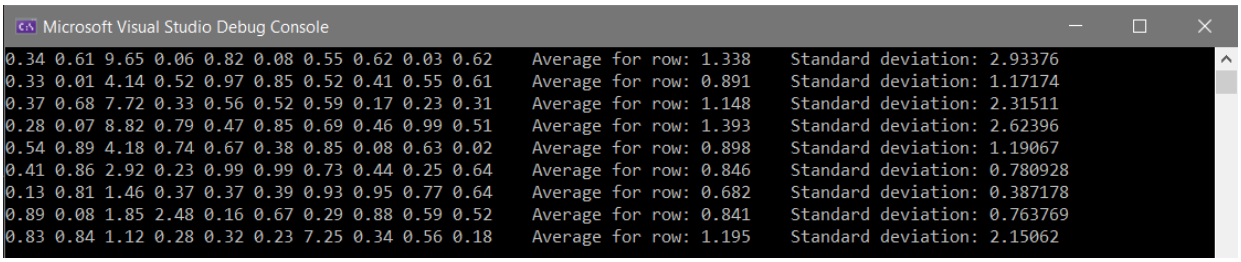

Рис. 3. Результати виконання програми із узгодженим/одночасним використанням потоків

Висновки

У сьогоднішніх реаліях, виробники програмних продуктів повинні чітко усвідомити, що для того щоб використовувати збільшення обчислювальної потужності новітніх процесорів, у програмах необхідно застосовувати багатопотокові обчислення. Для підтримки та організації паралельних обчислень, програмне забезпечення повинно мати можливість розподіляти робоче навантаження між багатьма процесорами наявними в системі [12].

Бібліотека <thread> надає широкі можливості для організації паралельних та узгоджених/одночасних обчислень. При цьому власне важливо розуміти різницю між паралельними та узгодженими/одночасними обчисленнями, для того щоб визначити логіку побудови програми і правильно використовувати функціонал, який пропонує бібліотека <thread>.

Список використаних джерел

1. Sutter H. (2009). The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. Retrieved from: www.gotw.ca/publications/concurrency-ddj.htm

2. Acar U. A., Charguéraud A., Rainey M. (2016). An Introduction to Parallel Computing in C++. Retrieved from: www.cs.cmu.edu/afs/cs/academic/class/15210-f18/www/pasl.html

3. Wataru E., Shigeyuki S., Kenjiro T. (2022). ComposableThreads: Rethinking User-level Threads with Composability and Parametricity in C++, Journal of Information Processing, Vol.30, 269–282. https://doi.org/10.2197/ipsjjip.30.269

4. Qiu W., Zhang Y., Wang L. (2019). Visual C + + and MFC Application in Safety Monitoring System of Airplane Depot, Advances in Computer Science Research, volume 88, 40-45, CNCI 2019, Atlantis Press. https://doi.org/10.2991/cnci-19.2019.6

5. Thelin R. (2020). A tutorial on modern multithreading and concurrency in C++. Retrieved from: www.educative.io/blog/modern-multithreading-and-concurrency-in-cpp

6. Сіциліцин Ю. О. (2022). Моделювання змісту дисципліни «Паралельні та розподілені обчислення», Педагогічні науки: теорія та практика, № 4 (44), 23-28. https://doi.org/10.26661/2786-5622-2022-4-03

7. Wilson C. (2019). Multithreading and concurrency fundamentals. Retrieved from: www.educative.io/blog/multithreading-and-concurrency-fundamentals

8. Williams A. (2019). C++ Concurrency in Action, Second Edition. Manning Publications Co. Shelter Island, NY.

9. The Difference Between Asynchronous and Multi-Threading (2023). Retrieved from: www.baeldung.com/cs/async-vs-multi-threading

10. MemarianK.,Matthiesen J., Lingard J.,Nienhuis K., Chisnall D., Watson R.N.M., Sewell P. (2016). Into the depths of C: elaborating the de facto standards, PLDI '16: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, 1–15.https://doi.org/10.1145/2908080.2908081

11. Mahapatra S. (2023). Multithreading in C++. Retrieved from: www.geeksforgeeks.org/multithreading-in-cpp

12. Ajkunic E., Fatkic H., Omerovic E., Talic K., Nosovic N. (2012). A Comparison of Five Parallel Programming Models for C++, 2012 Proceedings of the 35th International Convention, MIPRO 2012, Opatija, Croatia, May 21-25, 2012. IEEE 2012, 1780-1784.

Andriy Andrushko[1], Oleksandr Markelov[2]

[1]Computer Aided Design Department, Lviv Polytechnic National University,
12, St. Bandera str., Lviv, Ukraine, E-mail: andrii.m.andrushko@lpnu.ua, ORCID 0000-0003-4229-7589
[2]Computer Aided Design Department, Lviv Polytechnic National University,
12, St. Bandera str., Lviv, Ukraine, E-mail: oleksandr.e.markelov@lpnu.ua, ORCID 0000-0002-2432-0768

CREATING MULTITHREADED PTOGRAMS IN C++

Received: July 17, 2023 / Revised: September 28, 2023 / Accepted: October 10, 2023

Abstract. For many years, the increase in computing power of modern devices is achieved not by increasing the clock frequency and bandwidth of CPUs, but by using hyper-threaded and multi-core architectures. This simple change in approach to the CPU design led to dramatic changes in the organization of computing and became a turning point for software developers. Software that is going to take advantage of the increased computing power of multi-core architectures must be designed to be able to perform multiple tasks simultaneously. When covering the topic of parallel/simultaneous computing, scientists and IT professionals use two terms: (1) parallelism and (2) concurrency. A third important term when considering parallel/simultaneous computing is “multithreading”. In C++, the two most common ways to implement parallelism are concurrency and parallelism itself. Although they can be used in other programming languages, C++ stands out for its ability to use concurrent computations with lower-than-average utilization of general machine resources, as well as its ability to execute complex instructions. The C++11 standard introduced support for multi-threaded programs. The C++ standard recognized the existence of multithreading in the language and provided components for writing multithreaded programs in the <thread> library. This made it possible to write multithreaded programs in C++ without relying on platform-specific extensions and enabled the creation of portable multithreaded code with guaranteed behavior. The <thread> library provides extensive opportunities for organizing parallel and concurrent calculations, it also contains the C++ memory model, conditional variables, mutexes, etc. for thread synchronization. The article analyzes the possibilities of writing programs in the C++ using multiple threads for parallel and concurrent execution of tasks. It also considers the features of the C++ and the tools of the <thread> library that make parallelism and concurrency possible.

Key words: C++, multithreading, parallelism, concurrency, thread synchronization.