Интернет-журнал «Отходы и ресурсы» <https://resources.today>
Russian Journal of Resources, Conservation and Recycling

2020, №2, Том 7 / 2020, No 2, Vol 7 <https://resources.today/issue-2-2020.html>

URL статьи: <https://resources.today/PDF/09INOR220.pdf>

DOI: 10.15862/09INOR220 (<http://dx.doi.org/10.15862/09INOR220>)

Ссылка для цитирования этой статьи:

Зайнетдинов А.Р., Рудзейт О.Ю., Недяк А.В. Улучшение производительности фрагментов компьютерного кода с большими временными затратами на основе технологий параллельного программирования // Интернет-журнал «Отходы и ресурсы», 2020 №2, <https://resources.today/PDF/09INOR220.pdf> (доступ свободный). Загл. с экрана. Яз. рус., англ. DOI: 10.15862/09INOR220

For citation:

Zainetdinov A.R., Rudzeyt O.U., Nedyak A.V. (2020). Improving the performance of fragments of computer code with high time costs based on parallel programming technologies. *Russian Journal of Resources, Conservation and Recycling*, [online] 2(7). Available at: <https://resources.today/PDF/09INOR220.pdf> (in Russian) DOI: 10.15862/09INOR220

УДК 004.358

ГРНТИ 20.51.24

Зайнетдинов Артём Ришатович

ФГОУ ВО «Дальневосточный федеральный университет», Владивосток, Россия
Магистрант
E-mail: zainet.ar@gmail.com

Рудзейт Олег Юрьевич

ФГОУ ВО «Дальневосточный федеральный университет», Владивосток, Россия
Магистрант
E-mail: rudzeyt18@mail.ru

Недяк Антон Витальевич

ФГОУ ВО «Дальневосточный федеральный университет», Владивосток, Россия
Магистрант
E-mail: nedyaq@gmail.com

Улучшение производительности фрагментов компьютерного кода с большими временными затратами на основе технологий параллельного программирования

Аннотация. В данной статье рассмотрен процесс преобразования выполнения программы из однопоточного режима в многопоточный. Проведена оценка влияния количества потоков выполнения на общее время исполнения кода. Проблема затрат большого количества времени на выполнения программ решается с помощью инкрементального распараллеливания, когда в код, который предназначен для последовательного исполнения, добавляются распараллеливающие директивы. Приводятся аналоги библиотек для работы с потоками в других языках программирования и преимущества работы OpenMP над другими библиотеками. Рассмотрена модель работы распараллеливающих директив OpenMP, позволяющих без особых временных затрат выполнить фрагмент кода во многопоточном режиме. Также в статье представлен пример программного кода, реализованного с помощью языка C++, в который встроены распараллеливающие директивы библиотеки OpenMP. В данной модели оценено время исполнения фрагмента кода программы при различном количестве потоков. Исследование показало, что с увеличением количества потоков время выполнения программы

значительно уменьшается в среднем в 1,5–2 раза. Это происходит до того момента, пока количество потоков не станет равно количеству ядер процессора. Если количество потоков становится больше, чем количество ядер процессора, то время выполнения программы остается практически неизменным. Представленная библиотека OpenMP позволяет значительно сократить время работы программы, которое может быть критичным для работ систем реального времени, сократить время создания многопоточных приложений простыми вставками директив библиотеки, уменьшить количество ошибок, допускаемых программистом в ходе работы с многопоточными приложениями.

Ключевые слова: потоки; распараллеливание; библиотека OpenMP; язык программирования C++; многопоточность; производительность; директива

Введение

Компьютерные технологии в совокупности с мощным аппаратным обеспечением способны достаточно быстро исполнять программные инструкции в доли секунды, но с увеличением сложности программ время выполнения их увеличивается до секунд или десятков секунд, что может привести к задержке работы системы или привести к ее отказу.

Рассмотрение вопросов времени исполнения программного кода и его выполнения в многопоточном режиме занимает важное место в разработке ресурсоемких проектов. Применяемые на практике методы распараллеливания кода в различных языках программирования предоставляют такой эффект, как уменьшение времени работы программы, но являются достаточно трудозатратными для программиста. В языке программирования C# к проекту может быть подключена библиотека Threading [1], в языке Java – библиотека Thread [2], в языке Python – библиотека Thread [3], используя которую, на программиста возлагаются задачи по синхронизации, уничтожению и созданию потоков [4]. В процессе создания программ также требуется решить проблемы тупиков, которые могут сложиться из-за неправильного распараллеливания вычислительных процессов. В таких случаях, на поиск выхода из данной ситуации также требуется потратить большое количество времени.

Библиотека OpenMP является достаточно простым и легковесным инструментом, по сравнению с другими библиотеками, для распараллеливания вычислений и уменьшения времени исполнения программы, которая снимает нагрузку с программиста. Ее функционал достаточно прост в освоении благодаря директивам распараллеливания, которые встраиваются в код программы без фрагмента кода, который требует исполнения в многопоточном режиме, что уменьшает время на разработку проекта [5].

Модель работы с потоками в библиотеке OpenMP

Open MP – стандарт, который представляет описание совокупности директив компилятора и библиотечных процедур, предназначенных для воссоздания многопоточных программ на процессорах с большим количеством ядер и с общей доступной памятью [6].

При использовании многоядерных вычислительных систем с общей памятью предполагается, что имеющиеся процессоры обладают равной долей производительности, а также являются равноправными при доступе к общей памяти, время доступа к памяти является одинаковым при одновременном доступе некоторого числа процессоров к одному и тому же элементу памяти очередность (рисунок 1).

Рисунок 1. Архитектура систем с общей (разделяемой памятью) (составлено авторами)

Системы с общей памятью могут быть представлены в виде модели параллельного компьютера с произвольным доступом к памяти [7].

Обычный подход при организации вычислений для многоядерных вычислительных систем с общей памятью – это создание новых методов для распараллеливания выполнения последовательных программ, для которых компилятор автоматически, либо программист самостоятельно выделяет участки вычислений, независимых друг от друга [8–11].

Код программы, используя библиотеку OpenMP, начинается как единственный поток выполнения. Когда начальный поток встречает параллельную конструкцию, он создает новую группу потоков, состоящую из себя и некоторого числа дополнительных потоков, и становится главным в новой группе (рисунок 2).

Рисунок 2. Схема работы кода в многопоточном режиме (составлено авторами)

Все участники набора выполняют код в пределах параллельной конструкции. Когда поток заканчивает выполнение параллельной конструкции, он ожидает у неявного барьера в конце конструкции. Когда все члены набора доберутся до барьера, главный поток продолжит выполнение кода после окончания параллельных вычислений.

Постановка задачи

Для проверки эффективности работы библиотеки OpenMP и установления зависимости времени исполнения программного кода от количества параллельных потоков был написан фрагмент кода, на котором будут тестироваться директивы распараллеливания. Было предложено установить зависимость в работе алгоритма, который осуществляет проход по численному массиву, содержащем от 10 тыс. до 100 тыс. элементов и находит сумму произведений элементов данного массива (рисунок 3).

```
#include <conio.h>
#include <iostream>
#include <omp.h>
#include <ctime>
using namespace std;
double sum_arr( int *a, int n) {
    double val = 0;
    #pragma omp parallel shared(a) reduction (+: val) num_threads(18)
    {
        # pragma omp for
        for (int i = 0; i < n; i++)
            for(int j=0; j < n; j++)
                val += a[i]*a[j];
    }
    return val;
}
int main()
{
    const int n = 100000;
    int a[n];
    setlocale(LC_ALL, "Russian");
    for (int i=0; i < n; i++)
        a[i] = i;
    srand(time(0));
    cout << fixed << sum_arr(a,n) << endl;
    cout << "runtime = " << clock()/ 1000.0 << endl;
```

Рисунок 3. Пример распараллеливания последовательного кода (составлено авторами)

Директива Parallel определяет блок, в котором будут производиться параллельные вычисления [12].

Директива for разделяет работу цикла внутри параллельной области между потоками [12].

Директива shared определяет переменные, которые должны быть общими для всех потоков параллельной области [12].

Директива reduction обозначает переменную, с которой в цикле производится reduction-операция (например, суммирование). При выходе из цикла, данная операция производится над копиями переменной во всех потоках, и результат присваивается оригинальной переменной [13].

Директива NUM_THREADS определяет количество потоков в параллельной области [13].

Экспериментальные результаты

Тестирование библиотеки проводилось в диапазоне от 1 до 10 потоков для выявления такого количества потоков, при котором решение задачи займет наименьшее время, варьируя количество элементов в массиве от 10 тыс. до 100 тыс. элементов (таблица 1).

Таблица 1

Эффективность работы библиотеки в зависимости от количества потоков

№ опыта	Количество элементов	Количество потоков					
		1 поток	2 потока	4 потока	6 потоков	8 потоков	10 потоков
		t исп., с	t исп., с	t исп., с	t исп., с	t исп., с	t исп., с
1	10000	1,1	0,26	0,28	0,38	0,3	0,24
2	20000	1,29	0,8	0,67	0,65	0,7	0,66
3	30000	2,6	1,66	1,37	1,27	1,22	1,29
4	40000	4,5	2,95	2,23	2,15	2,23	2,12
5	50000	6,93	4,64	3,33	3,66	3,58	3,39
6	60000	9,9	6,39	5,03	4,7	4,68	4,79
7	70000	13,35	8,38	6,22	6,4	6,52	6,18
8	80000	17,35	10,77	7,9	8,35	8,26	7,93
9	90000	21,87	13,5	10,54	10	9,88	9,92
10	100000	27,08	16,35	13,06	12,24	12,28	12,32

Составлено авторами

Таблица 1 показывает, что вычисление суммы произведений элементов массива при любом количестве в нем элементов в одном потоке происходит в 2 раза дольше, чем при параллельной работе 4, 6, 8 и 10 потоков. Как можно наблюдать, прирост в производительности наблюдается уменьшении времени на 45 %. Для наглядности данные об эффективности работы библиотеки представлены на рисунке 3.

Рисунок 3. График зависимости времени исполнения программного кода от количества элементов массива (составлено авторами)

Далее в ходе эксперимента был взят отдельный массив с количеством элементов, равным 100 тыс., чтобы наглядно продемонстрировать работу библиотеки в рамках одного и того же массива, проход по которому осуществляется в количестве потоков от 1 до 18 (таблица 2).

Таблица 2

**Эффективность работы библиотеки в зависимости
от количества потоков в рамках прохода по одному массиву**

№ опыта	Количество потоков	t исп., с
1	1	27,07
2	2	16,62
3	4	12,87
4	6	12,24
5	8	12,07
6	10	12,26
7	12	12,38
8	14	12,26
9	16	11,93
10	18	12,24

Составлено авторами

При количестве потоков, которое больше 4 прирост в производительности почти не наблюдается, т. к. это связано с тем, что тестирование проводилось на процессоре, имеющем 4 вычислительных ядра, т. к. по началу при распараллеливании работы все ядра с одинаковой скоростью обрабатывают данные, но при превалировании числа потоков над числом ядер из-за недостатка вычислительной мощности происходит оптимизация времени простоя процессора. Например, при работе с двумя потоками, ядро переключается на второй поток, пока первый ожидает данные из памяти. При получении данных, ядро возвращается к работе с первым потоком, приостанавливая работу со вторым и т. д. [14–15]. График на рисунке 4, построенный на основе данных из таблицы 2 наглядно отражает эффективность работы библиотеки.

Рисунок 4. График зависимости времени исполнения программного кода от количества потоков для одного массива (составлено авторами)

Некоторые изменения в сторону увеличения во времени исполнения алгоритма на несколько сотых долей при увеличении количества потоков объясняется тем, что на эксперимент могут повлиять некоторые фоновые процессы компьютера и действия пользователя вплоть до движения указателя мышки по экрану.

Выводы

Библиотека OpenMP является очень простой легковесной и эффективной библиотекой для распараллеливания вычислений и не требует от программиста владения особыми навыками управления потоками. Также OpenMP позволяет значительно сократить время разработки какого – либо проекта, снимая нагрузку с программиста.

Скорость выполнения программы напрямую зависит от количества ядер процессора, превышая 4 потока выполнения производительность почти не растет из-за конфигурации вычислительного устройства, на котором проводился эксперимент. При увеличении количества ядер процессора можно достичь наиболее быстро исполнения кода программы.

Распараллеливание компьютерного кода с применением OpenMP дает прирост производительности до 50 % по времени выполнения.

На скорость исполнения кода может повлиять большое количество факторов: тип ядер, на которых исполняется программа, количество ядер, запущенные другие приложения, фоновые процессы и др.

ЛИТЕРАТУРА

1. Ferreira J.F., Sobral J.L. (2005) ParC#: Parallel Computing with C# in .Net. In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2005. Lecture Notes in Computer Science, Volume 3606, p. 226–238. DOI: 10.1007/11535294_20.
2. Ivannikov V.P., Avetisyan A.I., Gaissaryan S.S., Akopyan M.S. Implementation of Parallel Interpreter in the Development Environment ParJava. Programming and Computer Software. 2009. Volume 35, Issue 1. pp. 6–17. DOI: 10.1134/S0361768809010034.
3. Jon Kerr Nilsen, Xing Cai, Bjorn Hoyland, Hans Petter Langtangen, Simplifying Parallelization of Scientific Codes by a Function-Centric Approach in Python, Computational Science & Discovery, Volume 3(1), 2010. DOI: 10.1088/1749-4699/3/1/015003.
4. Клименко М.А., Трухан Д.А. Средства синхронизации потоков, критические секции и тупики // Современные наукоемкие технологии. – 2013. – № 8–2. – С. 200-202.
5. Клешнин В.Ю., Рыбаков. О применении технологий параллельного программирования для задач матричной алгебры в приложении к спектральному методу анализа, синтеза и идентификации систем управления // Математика и Математическое моделирование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2016. № 01. С. 1–27. DOI: 10.7463/mathm.0116.0841079.
6. Ханкин Константин Михайлович Сравнение эффективности технологий OpenMP, nVidia cuda и StarPU на примере задачи умножения матриц // Вестник ЮУрГУ. Серия: Компьютерные технологии, управление, радиоэлектроника. – 2013. – №1. – С. 36–38.
7. Г.Р. Эндрюс. Основы многопоточного параллельного и распределенного программирования. – М.: Изд. Дом Вильямс, 2003. – 330 с.
8. V. Cesare, I. Colonnelli, and M. Aldinucci, “Practical parallelization of scientific applications”, in Proc. of 28th euromicro intl. conference on parallel distributed and

- network-based processing (pdp), Västerås, Sweden, 2020, pp. 376–384. DOI: 10.1109/PDP50117.2020.00064.
9. Elber R., Ruymgaart A.P., Hess B., SHAKE parallelization, The European Physical journal. Special Topics, 01 Nov 2011, Volume 200(1), p. 211–223. DOI: 10.1140/epjst/e2011-01525-9.
 10. German Puebla, Manuel Hermenegildo, Abstract multiple specialization and its application to program parallelization, The Journal of Logic Programming, Volume 41, Issues 2–3, November–December 1999, Pages 279–316. DOI: 10.1016/S0743-1066(99)00031-X.
 11. Gustavsson A., Gustafsson J., Lisper B. (2014) Timing Analysis of Parallel Software Using Abstract Execution. In: McMillan K.L., Rival X. (eds) Verification, Model Checking, and Abstract Interpretation. VMCAI 2014. Lecture Notes in Computer Science, vol 8318. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-54013-4_4.
 12. Zhu, Weirong, Juan Del Cuvillo, and Guang R. Gao. Performance Characteristics of OpenMP Language Constructs on a Many-core-on-a-chip Architecture., OpenMP Shared Memory Parallel Programming. Springer Berlin Heidelberg, 2008. 230–241. DOI: 10.1007/978-3-540-68555-5_19.
 13. Sara Royuela, Roger Ferrer, Diego Caballero, Xavier Martorell, Compiler analysis for OpenMP tasks correctness. Proceedings of the 12th ACM International Conference on Computing Frontiers May 2015 Article No.: 7 Pages 1–8 <https://doi.org/10.1145/2742854.2742882>.
 14. Jin H., Jespersen D., Mehrotra P., Biswas R., Huang L., Chapman B. (2011) High performance computing using MPI and OpenMP on multi-core parallel systems. Parallel Comput 37(9), p. 562–575 Emerging Programming Paradigms for Large-Scale Scientific Computing. DOI: <https://doi.org/10.1016/j.parco.2011.02.002>.
 15. Theo Ungerer, Borut Robič, Jurij Šilc, Multithreaded Processors. The Computer Journal, Volume 45, Issue 3, 2002, Pages 320–348. DOI: <https://doi.org/10.1093/comjnl/45.3.320>.

Zainetdinov Artem Rishatovich

Far eastern federal university, Vladivostok, Russia
E-mail: zainet.ar@gmail.com

Rudzeyt Oleg Urievich

Far eastern federal university, Vladivostok, Russia
E-mail: rudzeyt18@mail.ru

Nedyak Anton Vitalievich

Far eastern federal university, Vladivostok, Russia
E-mail: nedyaq@gmail.com

Improving the performance of fragments of computer code with high time costs based on parallel programming technologies

Abstract. This article describes the process of converting the program from single-threaded to multithreaded mode. The influence of the number of threads of execution on the total execution time of the code. The problem of spending much time to run programs solved by incremental parallelization when the code is designed for sequential execution, are added to the parallelizing directives. Are similar libraries for working with threads in other programming languages and the advantages of OpenMP over other libraries. Considered a model of parallelizing OpenMP directives, which is time-consuming to run the snippet in multi-threaded mode. The article also presents example code implemented using the C++ language, which built the parallelizing directives of OpenMP. In this model, the estimated execution time of a code snippet of a program with different number of threads. The study showed that with increasing the number of threads the run time is significantly reduced on the average in 1,5–2 times. This happens until the moment when the number of threads will be equal to the number of processor cores. If the number of threads becomes greater than the number of processor cores, the execution time of the program remains virtually unchanged. Are the OpenMP library allows to significantly shorten the program, which can be critical for works of real-time systems, to reduce the time of creating multithreaded applications is simple inserts directives of the library, to reduce the number of errors made by the programmer in the course of working with multi-threaded applications.

Keywords: threads; parallelization; OpenMP library; C++ programming language; multithreading; performance; directive