УДК 004.428.4. 004.272.26

DOI: 10.18454/2313-1586.2017.01.127

Корниенко Андрей Викторович
кандидат технических наук,
научный сотрудник,
Горный Институт КНЦ РАН,
184209 г. Апатиты, Мурманская обл.,
ул. Ферсмана, 24
e-mail: kornienko@goi.kolasc.net.ru

ОПЫТ ПРИМЕНЕНИЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В АЛГОРИТМАХ СИСТЕМЫ MINEFRAME

Аннотация:

В статье рассмотрены различные способы организации параллельных вычислений с использованием вычислительных устройств, входящих в состав современных персональных компьютеров, предложен вариант организации и управления параллельными вычислениями, позволяющий увеличить производительность существующих алгоритмов с минимальными трудозатратами.

Ключевые слова: параллельные вычисления, система MineFrame

Kornienko Andrew V.
candidate of technical sciences, researcher,
The Mining Institute KSC RAS,
184209 Russia, Apatite, Murmansk region,
24 Fersman st.
e-mail: kornienko@goi.kolasc.net.ru

PARALLEL COMPUTING IN MINEFRAME SYSTEM ALGORITHMS

Abstract:

Different approaches of parallel computing on modern PC's computational devices are considered. The variant of parallel computing arrangement and management that permits to increase the existing algorithms capacity with minimum labor expenditures is proposed.

Key words: parallel computing, MineFramesoftware.

Горно-геологическая информационная система MineFrame [1] предназначена для решения геологических, маркшейдерских и технологических задач горного производства. Решение данных задач сопряжено со значительным объемом вычислений, что создает предпосылки для поиска способов ускорения работы алгоритмов с целью повышения производительности труда специалистов горных предприятий.

Повышение производительности вычислений достигается за счет алгоритмической оптимизации, а также за счет применения параллельной обработки данных в случаях, когда это возможно. Оптимизация алгоритма с целью снижения времени, необходимого для расчетов, предполагает в основном решение задачи за меньшее количество операций (высокоуровневая оптимизация), а также организацию вычислений с учетом архитектурных особенностей вычислительных платформ (низкоуровневая оптимизация). Параллельные же расчеты целесообразно применять в случаях, когда элементы данных могут быть обработаны независимо друг от друга либо эти зависимости незначительны. В противном случае основное время будет тратиться на синхронизацию доступа к общим ресурсам с применением того или иного вида блокировок, что в конечном итоге лишь увеличит время расчета.

В рамках данной статьи будут рассмотрены вопросы параллельной обработки данных, т. к. алгоритмическая оптимизация, несмотря на наличие общих принципов оптимизации, подразумевает в большинстве случаев «индивидуальный» подход к алгоритму, в то время как параллельные вычисления могут быть применены к целому ряду алгоритмов, в которых отсутствуют зависимости между обрабатываемыми элементами данных.

На сегодняшний день в сегменте персональных компьютеров (PC) устройствами, которые аппаратно поддерживают параллельные вычисления, являются центральный процессор (CPU) и графический ускоритель (GPU). Современный CPU является универсальным вычислительным устройством, архитектурно состоящим из нескольких физических (а также, возможно, логических) ядер. Назначением GPU является расчет и вывод

трехмерных графических данных с высокой скоростью, соответственно, данное устройство может быть использовано в ряде задач, схожих алгоритмически с обработкой 3D-графики. В подобных вычислениях GPU значительно превосходит CPU в производительности. В свою очередь CPU за счет своей универсальности предоставляет более широкие возможности для распараллеливания вычислений, в том числе и в сложных многовариантных алгоритмах, в которых использование GPU в большинстве случаев является нецелесообразным в силу весьма ограниченного функционала графических ускорителей.

Для использования возможностей GPU и CPU как многопоточных вычислительных устройств реализован набор программных интерфейсов, таких, например, как библиотеки OpenMP [2], OmniThreadLibrary [3], CUDA [4] и OpenCL [5]. Библиотеки OpenMP и OmniThreadLibrary предназначены для упрощения разработки программ на C++, Fortran и Delphi, использующих многопоточность на CPU. При применении CPU также имеется возможность использования соответствующего функционала операционной системы, предназначенного для запуска и выполнения параллельных потоков. Технология CUDA предоставляет возможность использования графических ускорителей NVidia в качестве вычислительных устройств для решения прикладных задач. Кроссплатформенная библиотека OpenCL позволяет использовать как CPU, так и GPU различных производителей совместно в рамках одного PC. При этом вычислительные устройства программно представляются как устройства с архитектурой, аналогичной GPU.

Следует отметить, что в системе MineFrame реализован целый ряд вычислительных алгоритмов, обрабатывающих значительные объемы данных. В каждом таком алгоритме уже реализованы решения для их оптимизации, включающие в себя, например, фильтрацию данных по различным критериям с последующим использованием получившихся выборок в дальнейших расчетах, оптимизацию поиска и т. д. Следовательно, при применении технологий параллельных вычислений к таким алгоритмам целесообразно сохранить все преимущества реализованных оптимизационных решений. Это позволяет сформулировать требования к перечисленным выше библиотекам, как к программному обеспечению (ПО) для организации параллельных вычислений в алгоритмах системы MineFrame:

1. Минимальные трудозатраты при многопоточной реализации алгоритма на базе уже реализованного и оптимизированного однопоточного варианта.
2. Сохранение общей структуры однопоточного алгоритма без изменений.
3. Максимально возможная производительность при соблюдении первых двух требований.

Рассмотрим более подробно возможности применения ПО для параллельных вычислений с учетом сформулированных требований. Библиотеки OpenMP и OmniThreadLibrary довольно полно отвечают требованиям 1 и 2, но в связи с многовариантным расчетом элемента данных возникает необходимость в динамическом распределении нагрузки между потоками во время выполнения. Отмеченные библиотеки имеют сравнительно простой вариант распределения нагрузки, не учитывающий в полной мере возможность многовариантной реализации, что делает их в большей степени ориентированными на распараллеливание на уровне задач, а не данных. В свою очередь CUDA и OpenCL предназначены для распараллеливания на уровне данных, и имеют соответствующие аппаратные реализации распределения нагрузки, но для их применения необходимо частично или полностью модифицировать структуру алгоритмов, что приводит к существенному увеличению трудозатрат. При этом выигрыш в производительности может оказаться незначительным.

Таким образом, для выполнения перечисленных выше требований параллельные вычисления целесообразно организовать на базе CPU с реализацией эффективного распределения нагрузки между потоками и ограничения или исключения доступа к общим

ресурсам из каждого потока. Следует отметить, что наиболее часто используемым разделяемым ресурсом, как правило, является память, выделение/освобождение которой осуществляется через менеджер памяти. При этом практически любая потокобезопасная реализация такого менеджера подразумевает наличие межпоточной блокировки как при выделении, так и при освобождении блоков памяти. Следовательно, для достижения максимальной производительности необходимо сводить к минимуму обращения к менеджеру памяти из параллельно работающих потоков. С другой стороны, такой метод оптимизации, как фильтрация данных, подразумевает формирование вспомогательных динамических массивов, что неизбежно приводит к выделению/освобождению блоков памяти неизвестного заранее размера. Выходом в данном случае может являться повторное использование памяти в каждом потоке, а для случаев динамических массивов или списков – только выделение памяти без ее освобождения до конца расчета (очистка списка в потоке подразумевает в данном случае лишь сброс счетчика элементов без освобождения занимаемой памяти).

С целью выполнения перечисленных выше требований было разработано два программных механизма на базе WindowsAPI, предназначенных для организации параллельных вычислений поверх существующих однопоточных реализаций алгоритмов. При использовании каждого из этих механизмов подразумевается применение оптимальных методов работы с менеджером памяти.

В соответствии с первым механизмом каждый элемент данных обрабатывался в отдельном потоке, по завершению расчета ресурсы потока освобождались, и для обработки следующего элемента создавался новый поток. Количество одновременно работающих потоков определялось количеством ядер CPU. Схема работы данного механизма представлена на рис. 1. Преимуществом данного метода организации вычислений являлась простота реализации и автоматическое распределение нагрузки, т. к. один поток обрабатывал не группу элементов данных (в результате чего потоки могли завершаться через разные промежутки времени, что привело бы к снижению производительности), а всего один элемент. Еще одним преимуществом данного метода является простота обработки элементов данных, организованных в различные структуры (деревья, динамические массивы различной размерности), т. к. основной цикл обхода элементов абсолютно идентичен однопоточному варианту. Как следствие –отсутствие необходимости во вспомогательном массиве для промежуточного хранения ссылок на обрабатываемые элементы.

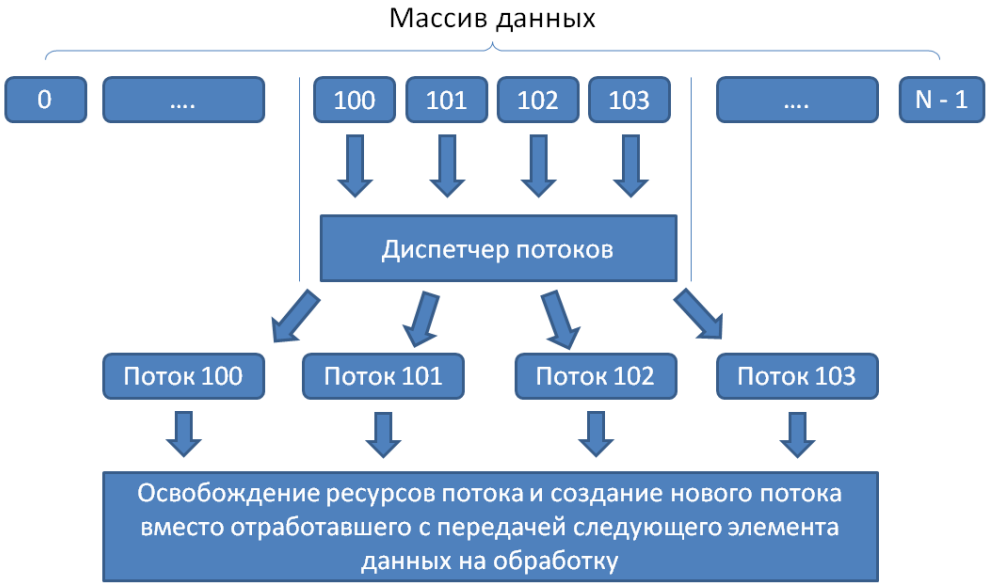

Рис. 1 – Организация параллельных вычислений с динамическим созданием потоков

В результате внедрения данного механизма в ряде алгоритмов удавалось сократить время расчета в N раз, где N – количество физических ядер процессора. Но по результатам практического применения проявился и недостаток данного метода: в случаях, когда время обработки элемента данных незначительно, использовать все ядра CPU не представляется возможным, производительность при этом существенно падает. Связано это с особенностью организации работы потоков, на каждый из которых выделяется определенный квант времени в операционной системе. В случае, когда вычисления в потоке завершаются раньше этого времени, возникает простой, и при передаче управления главному потоку-диспетчеру последний создаст новый поток для обработки следующего элемента данных, даже если загружены не все ядра CPU. В таком случае значительная часть времени работы будет отведена созданию/удалению потоков и простоям, что на практике увеличивало время расчета даже по сравнению с однопоточным вариантом.

В связи с тем, что изложенный выше механизм не является универсальным и не обеспечивает увеличения производительности при любых реализациях обработки элементов данных, был разработан механизм, схожий со схемой организации вычислений в OpenCL и CUDA, позволяющий при этом использовать все функциональные возможности CPU. Схема работы данного механизма представлена на рис. 2.

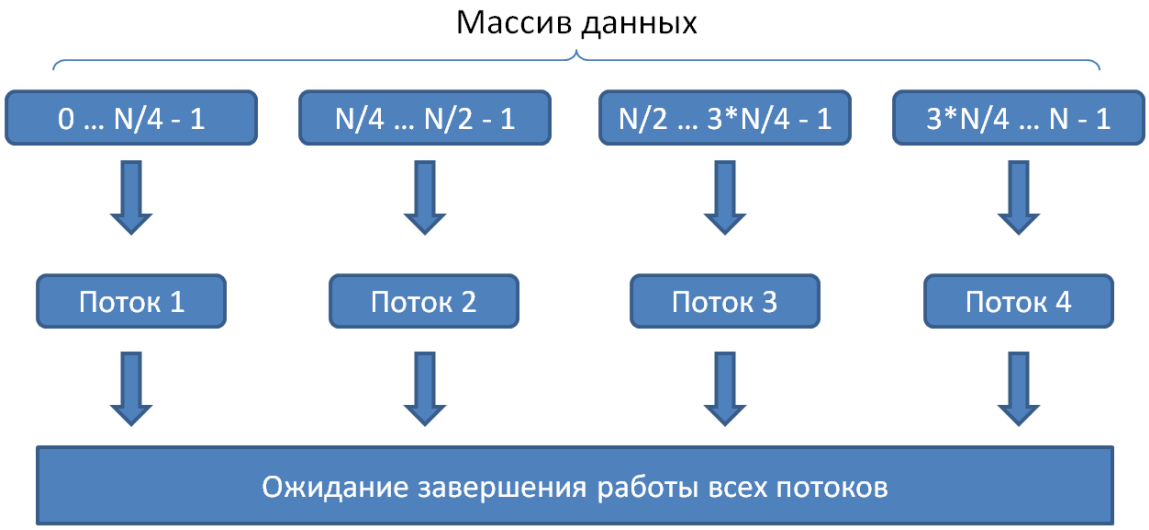

Рис. 2 – Организация параллельных вычислений
с формированием диапазонов данных

Суть работы механизма в следующем: данные передаются на обработку в виде вспомогательного массива ссылок на элементы данных, и в пределах этого массива определяются диапазоны индексов элементов, обрабатываемых в конкретном потоке. Количество работающих одновременно потоков соответствует количеству ядер CPU, и все они запускаются с учетом сформированных диапазонов данных, т. е. каждый поток обрабатывает часть элементов, являющихся подмножеством исходного массива. В данной схеме сведено к минимуму создание/удаление потоков, но присутствует вспомогательный массив ссылок на обрабатываемые элементы. К тому же в данном случае лишь частично реализуется распределение нагрузки, которое может оказаться неоптимальным при формировании исходных диапазонов данных.

Для достижения максимальной производительности был реализован функционал для динамического распределения нагрузки между потоками, схема которого приведена на рис. 3. Суть распределения заключается в следующем: в случае, если один из потоков обработал свой блок данных раньше других, оставшиеся потоки завершают работу, обработав предварительно элемент в текущей итерации. В результате при передаче управления главному потоку последний будет обладать информацией о том, какие диапазоны

данных не были обработаны, и сформирует на их основе новые диапазоны в соответствии с изначальным количеством потоков. Далее потоки снова запускаются с новыми диапазонами данных, а сам цикл продолжается до тех пор, пока не будут обработаны все элементы.

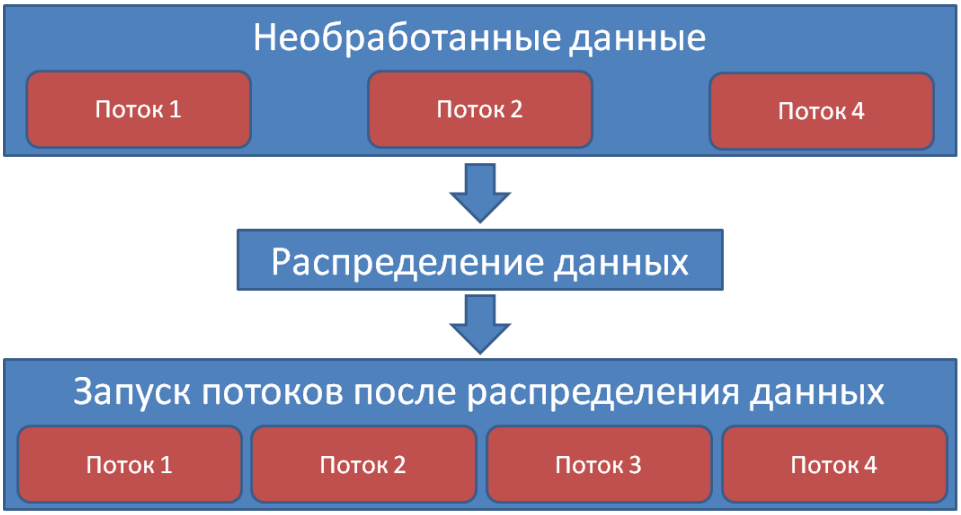

Рис. 3 – Динамическое распределение нагрузки между потоками

Практическое применение данного механизма показало стабильный прирост производительности вне зависимости от реализации обработки элементов данных (время расчета сокращалось в N раз, где N – количество физических ядер процессора).

Таким образом, предложен и апробирован простой в реализации и применении метод повышения производительности прикладных алгоритмов, допускающих параллельную обработку данных, с минимальными трудозатратами.

Дальнейшее развитие предложенного механизма может заключаться в разработке и реализации интерфейса для управления обходом элементов, организованных в произвольные структуры данных, что позволит исключить вспомогательный массив ссылок на обрабатываемые элементы.

Литература

1. Наговицын О.В. Автоматизированные инструменты инженерного обеспечения горных работ в системе MINEFRAME / О.В. Наговицын, С.В. Лукичёв // ГИАБ. – 2013. – № 7. – С. 184 - 192.

2. Официальный сайт разработчиков OpenMP [Электронный ресурс] - Режим доступа: http://openmp.org

3. Официальный сайт разработчиков OmniThreadLibrary [Электронный ресурс] – Режим доступа: http://omnithreadlibrary.com

4. Официальный сайт NVidia [Электронный ресурс] – Режим доступа: http://www.nvidia.ru/object/cuda-parallel-computing-ru.html

5. Официальный сайт разработчиков стандарта OpenCL [Электронный ресурс] – Режим доступа: https://www.khronos.org/opencl