УДК 004.42

Обзор сред для реализации параллельного программирования

Козич Виталий Геннадьевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Васильева Полина Александровна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье сделан обзор сред для реализации параллельного программирования. Приведены их достоинства и недостатки.
Ключевые слова: параллельное программирование, параллельные вычисления, OpenMP, MPI, Java.

Overview of environments for implementing parallel programming

Kozich Vitaliy Gennadievich
Sholom-Aleichem Priamursky State University
Student

Vasilyeva Polina Alexandrovna
Sholom-Aleichem Priamursky State University
Student

Abstract
This article provides an overview of the environments for implementing parallel programming. Their advantages and disadvantages are given.
Keywords: parallel programming, parallel computing, OpenMP, MPI, Java.

Параллельные вычисления — компьютерный способ организации вычислений, при котором программное обеспечение создается как набор взаимосвязанных одновременных вычислительных процессов. Достаточно актуальный термин, который включает в себя параллелизм в программировании и разработку эффективно действующих аппаратных средств. Раньше параллельным программированием занимались только разработчики-одиночки, заинтересованные задачами для суперкомпьютеров. Но на сегодняшний день, когда многоядерные процессоры стали поддерживать обычные приложения, технология параллельного программирования быстро стала на столько актуальной, что ее должен освоить любой профессиональный разработчик программного обеспечения.

Задача выбора определенной среды для параллельного программирования может оказаться очень сложной. Все эти системы непросты в изучении и требуют для этого достаточно большого промежутка времени. Из этого следует, что для выбора самой подходящей осваивать все нет смысла. Для программиста нужна лишь возможность изучить в быстрые сроки различные среды и ознакомиться с их основными особенностями на уровне для того самого выбора определенной системы, чтобы дальше детально изучить ее.

В данном исследовании сделан общий обзор на самые популярные среды для реализации параллельного программирования, подробно описаны основные способы их применения, а также плюсы и минусы систем: OpenMP, MPI и Java.

В качестве основы для данного исследования использовались книги А.С. Антонова про реализацию параллельного программирования в OpenMP и MPI [1-2]. Многопоточность Java была рассмотрена одним из пользователей ресурса «Хабрахабр» [3]. Про достоинства и недостатки параллельного программирования опубликовали статью Козич В.Г. и Бондаренко В.В. [4].

OpenMP

OpenMP (Open Multi-Processing) – это API-интерфейс, являющийся открытым стандартом разработки параллельных приложений для компьютеров, которые используют совместную память. Основной задачей является упрощение программирования, ориентированного на циклы. Разрабатываемые программы в основном создаются для высокопроизводительных вычислений. Также, в OpenMP присутствуют компоненты поддержки алгоритмов для распараллеливания: «главный и рабочий процесс», SPMD и конвейерный. Конечно, есть и другие.

OpenMP на данный момент является успешным языком параллельного программирования. Он есть на каждом компьютере, который использует совместную память. Также, относительно недавно был разработан вариант с поддержкой кластеров. OpenMP использует стиль программирования, при котором процесс распараллеливания достигается постепенно, пока имеющаяся линейная программа не превращается в параллельную. Этот плюс является также и минусом. Если, так называемое, превращение происходит постепенно, то разработчику тяжело справиться с широкомасштабной модернизацией программы, которой необходимо заниматься для достижений высокой производительности.

Основой для OpenMP является модель fork-join (разветвление-объединение). Разработка начинается с одного потока, далее, когда нужно распараллелить программу, выполняется разбиение на несколько потоков одной группы. Они выполняются параллельно в рамках одного фрагмента программного кода, который имеет название параллельный участок. В самом конце этого участка все потоки завершают свою работу и снова объединяются. После этого исходный поток продолжает выполнять работу

до тех самых пор, пока следующий параллельный участок не положит начало (или пока вся программа не завершится) (см. рис.1).

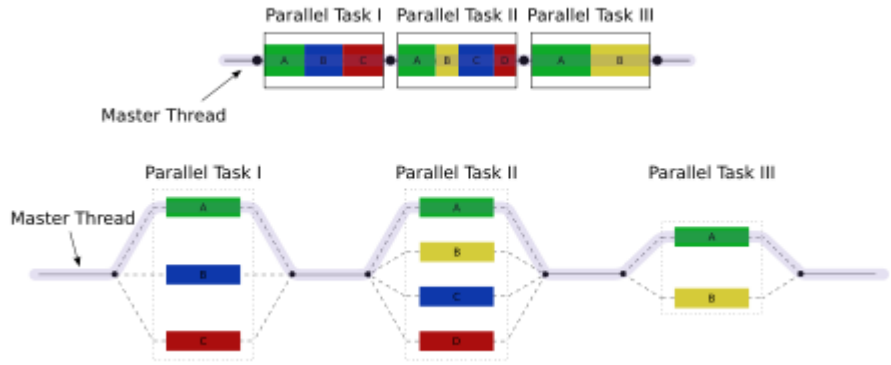

Рисунок 1. Механизм программирования «разветвление-объединение»

При создании потока и присвоении ему определенной задачи, программисту необходимо указать нужное действие. Следовательно, тот широкий спектр условий и конструкций, включенный в данную среду, разработчику необходимо знать. Но, есть и достоинство, часть работы возможно выполнить, имея в помощь небольшое подмножество всего языка.

MPI

MPI (Message Passing Interface) – интерфейс передачи сообщений, является одним из самых старых API-интерфейсов для параллельного программирования, который применяется и на сегодняшний день. Данная среда состоит из набора независимых процессов, взаимодействующих между собой путем отправления и приема сообщений. Огромным плюсом MPI являются достаточно низкие требования интерфейса к аппаратной части параллельного компьютера. Единственное что нужно интерфейсу это использование одной сети ядром и процессором, пригодных для обмена сообщений между двумя процессами. Это дает преимущество в работе MPI с любой стандартной параллельной системой (см. рис. 2).

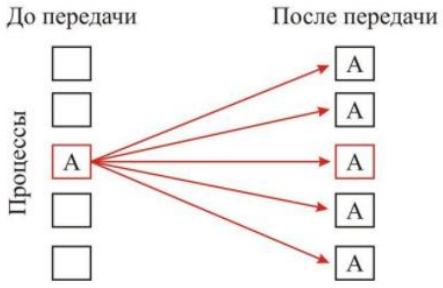

Рисунок 2. Механизим распараллеливания в MPI

Интерфейс MPI был разработан в последнем десятилетии прошлого века, когда кластеры только начали обретать популярность, а процессоры с высоким уровнем параллелизма (MPP) были доминантами высокопроизводительных вычислений. Разработчики MPP использовали

свою собственную систему для передачи сообщений. Производителей очень устраивал данный подход, так как пользователи вынуждены были привязаться к их линейке продуктов. Из-за отсутствия переносимой системы обозначений, программисты при смене компьютера прилагали колоссальное количество усилий для переноса своих приложений из одной системы передачи сообщений в совершенно другую.

Совершенно очевидно, что Интерфейс MPI не являлся первой библиотекой для переноса системы передачи сообщений, но он был первым разработанным с участием производителей академических институтов и национальных лабораторий. Естественно, он в скором времени стал стандартом для передачи сообщений, так как при его разработке приняли участие все ключевые разработчики. На данный момент, спустя три десятилетия, интерфейс остается самой распространенной средой обозначения параллельного программирования в рамках высокопроизводительных вычислений.

Основой разработки была поддержка широчайшего спектра аппаратных средств, а также сложнейших программных архитектур с точной модульной структурой. Ключевой термин, связанный с MPI – это коммуникатор. При создании набора различных процессов он образуют группу, которая совместно использует одну среду для связи.

Уникальный коммуникатор образуется из сочетания среды связи и группы процессов. Достоинства такого механизма становятся очевидными, если рассмотреть библиотеки и их использование в ПО. Конфликты сообщений разработчика библиотеки могут возникнуть в программе с сообщениями, использующими эту библиотеку из-за невнимательности программиста. Данная проблема решается наличием коммуникаторов. Они дают возможность создать собственную среду для связи, а также гарантию, что все что происходит в рамках этой библиотеки, не выйдет за ее границы.

Java

Язык Java был изначально разработан с интегрированной поддержкой мультипотоков. Они являются ключевыми в технологии Java: поддерживаются на языковом и виртуальном уровне, а также на уровне библиотек классов. В большинстве случаев потоки Java почти неотличимы от потоков pthreads. Библиотеки классов Java предоставляют доступ к классу thread, поддерживаемый широким набором механизмов для запуска, выполнения, приостановки и проверки состояния потока.

В каждую единицу времени одному ядру соответствует одна единица исполнения. То есть процессор с одним ядром может обрабатывать команды последовательно, один процесс за раз. Но запустить несколько параллельных потоков возможно и в системе с одноядерным процессором. В данном случае будет периодическое переключение потоков, выполняться которые будут по очереди. Такой механизм имеет название псевдо-параллелизм. Идет сохранение состояние каждого потока перед тем как будет совершено переключение на другой, далее система восстанавливает его для возвращения к выполнению потока. Состояние потока включает значения: стек, набор

значений регистров процессора, адрес исполняемой команды и другие (см. рис. 3).

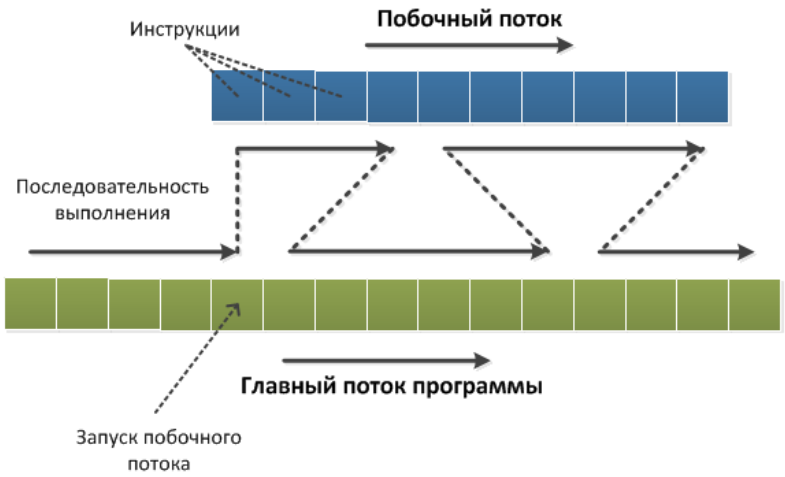

Рисунок 3. Схема псевдо-параллельного программирования

Поддержка потоков на основе переменных условий и мониторов относит к себе сложные наборы примитивов синхронизации. На языковом уровне синхронизированные методы внутри класса или блока кода не хотят выполняться параллельно. Эти методы и блоки выполняются благодаря мониторам, которые дают возможность убедиться, что доступ к данным присутствует, если они в согласованном состоянии. Мониторы, которые даны каждому объекту Java, работают почти так же, как пара переменного условия и мьютекса (взаимное исключение), определенных в pthreads. Но, разница от pthreads состоит в том, что поток Java возможно прервать, пока он имеет статус ожидания.

В тот момент, когда «чистые» потоки Java, описанные выше, являются самым нижним уровнем поддержки мультипотоковости, еще существуют различные высокоуровневые библиотеки параллелизма, которые нужны для расширения исходных базовых возможностей. В качестве примера можно привести пакет java.util.concurrent, включающий в себя большое количество модификаций базового распараллеливания: элементарные переменные, поддержка пулов потоков и сложные примитивы синхронизации. Увы, но некоторые из компонентов пакета util.concurrent не попали в стандарт J2SE и на сегодняшний день доступны в качестве отдельной библиотеки EDU.oswego.cs.dl.util.concurrent.

Одним из самых важнейших компонентов является FJTask, который реализует механизм распараллеливания «разветвление-объединение», о котором шла речь выше в обзоре OpenMP. Этот компонент предназначен, в главную очередь, для распараллеливания очень сложных вычислений, например, интегрирование чисел или умножение матриц. Цели FJTask – это облегченный вариант потоков Thread. Эту разработку наиболее часто именуют «параллелизм на основе задач». Задачи FJTask чаще всего реализуются при помощи того же самого пула потоков Java. Эти задачи дают

вариации самых обобщенных методов класса Thread: start(), yield() и join(). Основное достоинство в использовании задач FJTask получается за счет того, что они не имеют возможности поддержки блокировки операций разного рода. Задачи FJTask не были предназначены в качестве поддержки хаотичной синхронизации, так как нет способа приостановить, и далее продолжить выполнение отдельных задач после из запуска. Также, время на выполнение задач FJTask обязано быть ограниченным, и сама задача не должна включать в себя бесконечные циклы. Они должны просто исполняться до конца без всякого рода ожиданий и блокировки ввода-вывода. Задачи FJTask и потоки Thread имеют достаточно важные различия. Первые могут выполняться на три порядка быстрее, чем вторые, конечно же если они выполняют работу на виртуальных машинах Java с высокопроизводительным сбором мусора, который без исключений производит каждая задача FJTask, и отличной интегрированной поддержкой потоков.

В качестве заключения можно сделать вывод. Был сделан обзор на большой спектр наиболее популярных используемых сред для реализации технологии параллельного программирования. Все эти системы имеют различие в сложности, в количестве изменений, которые требуется внести в базовую линейную программу, и огромном объеме освоения. Эти важнейшие факторы нужно рассматривать с учетом предполагаемых для использования типов алгоритмов параллельного программирования. Данная статья может быть полезна студентам по дисциплинам в изучении программирования.

Библиографический список

1. Антонов А.С. Параллельное программирование с использованием технологии OpenMP. М.: МГУ, 2009. 77 с.
2. Антонов А.С. Параллельное программирование с использованием технологии MPI. М.: МГУ, 2004. 71 с.
3. Многопоточность в Java // Хабрахабр URL: https://habrahabr.ru/post/164487/ (дата обращения: 28.01.2018).
4. Бондаренко В.В., Козич В.Г., Плахотная Л.А. Достоинства и недостатки параллельного программирования // Современные научные исследования и инновации. 2016. №6. С. 204-208.