Тема работы:

Анализ эффективности синхронизации потоков в гибридных параллельных программах MPI+Threads

Состав коллектива:

1. Пазников Алексей Александрович, к.т.н., с.н.с. СПбГЭТУ «ЛЭТИ», руководитель
2. Юнгю Чен, магистрант. СПбГЭТУ «ЛЭТИ», исполнитель
3. Цянь Синьюй, магистрант. СПбГЭТУ «ЛЭТИ», исполнитель

Информация о гранте:

РНФ, проект № 22-21-00686 «Алгоритмы и программные средства оптимизации выполнения параллельных программ в модели удаленного доступа к памяти», руководитель – Пазников А.А., 2022-2023

Научное содержание работы:

1. Постановка задачи:

Цель данной научно-исследовательской работы состоит в изучении и применении гибридного метода параллельного программирования MPI+threads. Основной упор делается на реализацию коммуникации в многопоточном MPI и вычислениях над матрицами. В рамках исследования планируется измерение времени выполнения программы, сравнение производительности различных методов оптимизации и стратегий синхронизации потоков, а также проведение анализа экспериментальных результатов. Цель состоит в максимизации производительности современных компьютерных систем, сокращении накладных расходов на коммуникацию и повышении вычислительной эффективности в области высокопроизводительных вычислений.

2. Современное состояние проблемы:

Рассмотрим работы в области создания средств поддержки гибридных MPI+threads-программ.

В [1] предложен алгоритм вычисления целочисленного идентификатора контекста для создания нового MPI-коммуникатора в режиме MPI_THREAD_MULTIPLE. Алгоритм модифицирует процедуру выделения идентификатора для каждого процесса на основе глобальной структуры и операции MPI_Allreduce.

В [2] описаны четыре подхода к синхронизации потоков в MPI+threads-программах: Global, Brief Global, Per Object и Lock-free. Каждый подход предлагает разные методы блокировки и организации критических секций.

В [3] показано, что существующие реализации режима MPI_THREAD_MULTIPLE используют несправедливые алгоритмы синхронизации потоков. Предложены два алгоритма справедливой синхронизации, учитывающие особенности архитектуры NUMA.

В [4] проведен анализ справедливости захвата мьютексов Pthread в гибридной модели MPI+threads. Предложена улучшенная версия алгоритма блокировки потоков на основе Ticket Lock и алгоритм с приоритетом на основе CLH-LPW.

В [5, 6] предложены методы оптимизации гибридных MPI+threads программ, включая потокобезопасную хеш-таблицу, оптимизацию планирования потоков и изменение пула пакетов сообщений.

В [7] предложены два алгоритма оптимизации синхронизации потоков, уменьшающие количество ожидающих потоков и улучшающие выбор потока для выполнения операции.

В [8] представлен подход HDOT, основанный на разложении сложной задачи на предметно-иерархическую структуру задач, позволяющую выполнять операции на уровне потоков и уменьшить накладные расходы.

Описанные работы исследуют различные аспекты создания средств поддержки гибридных MPI+threads-программ, включая синхронизацию потоков, оптимизацию выполнения и использование предметно-иерархической декомпозиции задач.

1. Gropp W., Thakur R. Thread-safety in an MPI implementation: Requirements and analysis // Parallel Computing. 2007. V. 33. №. 9. pp. 595-604.

2. Balaji P. et al. Fine-grained multithreading support for hybrid threaded MPI programming //The International Journal of High-Performance Computing Applications. 2010. V. 24. №. 1. pp. 49-57.

3. Amer A. et al. MPI+ threads: Runtime contention and remedies //ACM SIGPLAN Notices. 2015. V. 50. №. 8. pp. 239-248.

4. Amer A. et al. Locking aspects in multithreaded MPI implementations //Argonne National Lab., Tech. Rep. P6005-0516. 2016.

5. Dang H. V., Snir M., Gropp W. Towards millions of communicating threads //Proceedings of the 23rd European MPI Users' Group Meeting. ACM, 2016. pp. 1-14.

6. Dang H. V., Snir M., Gropp W. Eliminating contention bottlenecks in multithreaded MPI //Parallel Computing. – 2017. – Т. 69. – С. 1-23.

7. Dang H. V. et al. Advanced thread synchronization for multithreaded MPI implementations // Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on. IEEE, 2017. pp. 314-324.

8. Ciesko J. et al. HDOT—An approach towards productive programming of hybrid applications //Journal of Parallel and Distributed Computing. – 2020. – Т. 137. – С. 104-118.

3. Подробное описание работы, включая используемые алгоритмы:

MPI поддерживает несколько уровней безопасности потоков, как показано на рисунке 19 ниже:

Support Levels	Description
MPI_THREAD_SINGLE	Only one thread will execute.
MPI_THREAD_FUNNELED	Process may be multi-threaded, but only main thread will make MPI calls (calls are ”funneled" to main thread). “Default”
MPI_THREAD_SERIALIZE	Process may be multi-threaded, any thread can make MPI calls, but threads cannot execute MPI calls concurrently (MPI calls are ”serialized").
MPI_THREAD_MULTIPLE	Multiple threads may call MPI, no restrictions.

Рис. 1 – Уровни потокобезопаности в MPI

Для реализации многопоточной связи MPI необходимо указать требуемый уровень безопасности потоков при инициализации MPI. Как показано в следующем коде:

В данной статье используется уровень потока MPI "MPI_THREAD_MULTIPLE", позволяющий неограниченно вызывать функции MPI из любого потока в каждом процессе. Гибридная модель параллельного программирования объединяет потоки и MPI для оптимальной производительности. Она позволяет группе потоков выполнять вычисления, а другой группе потоков обрабатывать коммуникацию, обеспечивая перекрытие и улучшение производительности. Многопоточность также улучшает параллелизм коммуникации в MPI. Гибридная модель объединяет преимущества общей и распределенной памяти, обеспечивая лучшую масштабируемость, упрощение кода, более эффективное использование кэшей и снижение необходимости в отдельном ранге для координации коммуникации. Гибридная модель взаимодействует между процессорами посредством обмена сообщениями MPI и использует общую память внутри узлов для параллелизма (рис. 2).

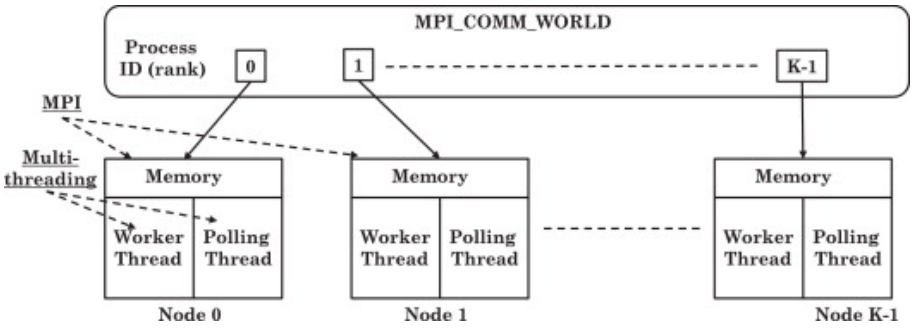

Рисунок 2 – Гибридная модель MPI

Эта модель гибридного программирования в наибольшей степени соответствует архитектуре кластера SMP и может в полной мере раскрыть потенциал аппаратной архитектуры. На рис. 3 показано, как работают потоки в гибридной модели (коммуникационная часть):

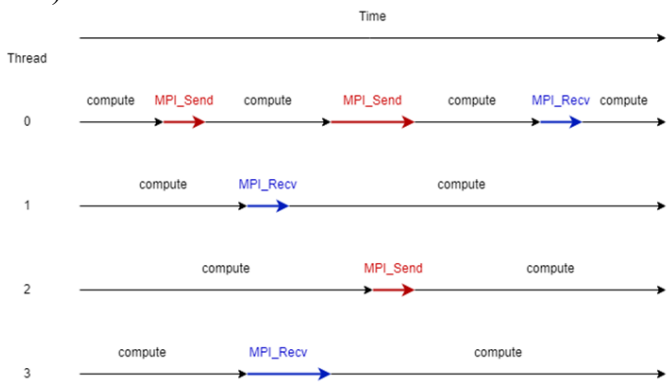

Рисунок 3 – Связь MPI при многопоточности

Эксперименты в этой статье были проведены в кластерной среде, предоставленной NSU. В кластере находится общее количество 48 двойных серверов высокой плотности HP BL2x220c G7 (2011), каждый из которых содержит две материнские платы, оснащенные собственной памятью и процессором. Сервер HP BL2x220c G7 имеет двухузловую архитектуру с двухканальной связью, при этом каждый узел имеет собственную память и процессор.

В гибридной модели параллельного программирования в каждом процессе MPI можно использовать OpenMP или pthread для достижения детализированной параллельности и ускорения вычислений умножения матриц. Это включает разделение задачи умножения подматрицы в каждом процессе на более мелкие подзадачи и их параллельное выполнение на нескольких потоках внутри процесса.

Для изучения различий в производительности умножения матриц при параллельном выполнении через процессы и потоки на одном узле будут использованы отдельно библиотеки MPI, OpenMP и Pthreads.

В рамках чистого MPI (параллельность через процессы) строки матрицы A разделены между разными процессами MPI, а матрица B передается всем процессам, чтобы каждый процесс ответственно вычислял часть строк результирующей матрицы C. Затем с использованием функции MPI_Gather результаты собираются и синхронизируются в корневом процессе. Потоковая параллельность реализуется с использованием библиотек OpenMP и Pthreads, что отражается в синхронной работе между потоками. Каждый поток отвечает за вычисление части результата умножения матрицы. Распределение задач между потоками обеспечивает равномерное распределение нагрузки в зависимости от числа строк матрицы.

Записывается и сравнивается время выполнения каждой программы при различных уровнях детализации, результаты показаны на рисунке 3.

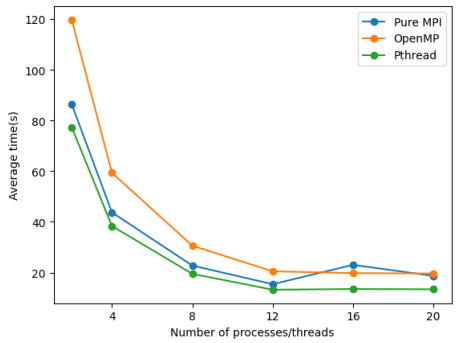

Рисунок 3 – Производительность параллельных процессов и потоков

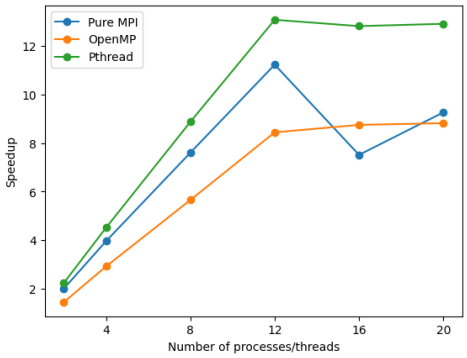

Рисунок 3 – Коэффициент ускорения процесса и потока

Исходя из вышеприведенных результатов, мы можем интуитивно видеть, что библиотека Pthreads, как легковесная и высокопроизводительная, обладает параллельной производительностью, превосходящей процессы MPI и OpenMP.

Также процессы MPI показывают хорошую производительность на грубозернистом уровне, даже лучшую, чем OpenMP.

Однако на мелкозернистом уровне производительность синхронизации потоков лучше.

MPI кольцевая коммуникация представляет собой передачу информации через MPI-коммуникацию между процессами в структуре кольца. В этом режиме каждый процесс является и отправителем, и получателем. Каждый процесс отправляет сообщение процессу слева и получает сообщение от процесса справа.

Для кольцевой коммуникации обычно используются функции MPI_Send и MPI_Recv для точечной коммуникации. Отправляющий процесс использует MPI_Send для отправки сообщений процессу слева, а принимающий процесс использует MPI_Recv для получения сообщений от процесса справа (см. рисунок 4).

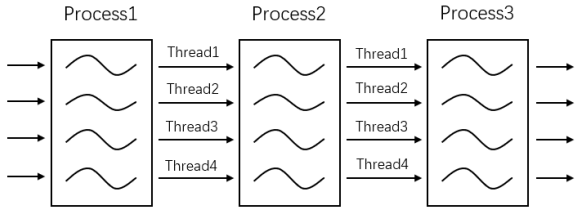

Рисунок 4 – Многопоточная кольцевая связь

Мы создаем четыре потока в каждом процессе, отправляем сообщения следующим потокам и получаем сообщения от предыдущих потоков. Затем записываем время коммуникации с помощью MPI_Wtime и подсчитываем суммарное время с использованием MPI_Reduce. Результат представлен на рисунке 26.

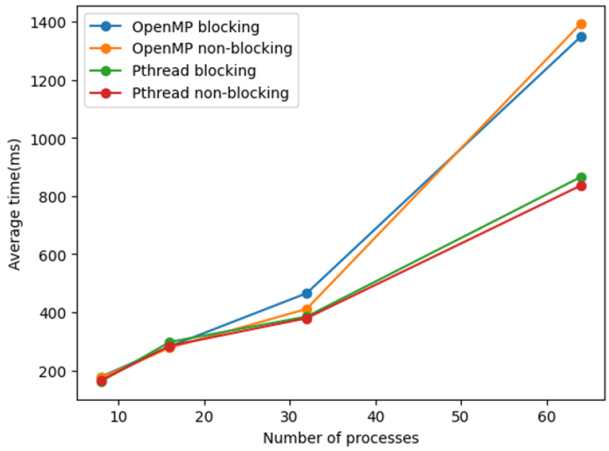

Рисунок 5 – Производительность параллелизма потоков при кольцевой связи

Из результатов на рисунке 26 видно, что с увеличением числа процессов, то есть увеличением числа коммуникаций между потоками, Pthreads обладает все более явными преимуществами в синхронизации по сравнению с OpenMP.

Однако при данной модели кольцевой коммуникации производительность блокирующего и неблокирующего режимов MPI практически неотличима.

Линейная рассылка - основной режим коммуникации в MPI. Корневой процесс отправляет одно сообщение всем остальным процессам. Линейная рассылка проста и надежна, но количество коммуникаций растет с числом процессов. В больших системах это может ограничить производительность. Однако, линейная рассылка подходит для многих задач из-за своей простоты и надежности.

Мы создаем 4 потока в корневом процессе и параллельно отправляем массив размером 3000*3000 другим процессам. Потоки выполняют блокирующую коммуникацию. Записываем время завершения отправки данных в основном процессе и получаем результат, показанный на рисунке 6.

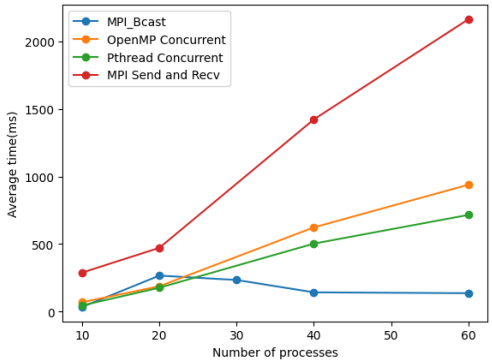

Рисунок 6 – Производительность вещания с линейной блокировкой MPI при многопоточной синхронизации

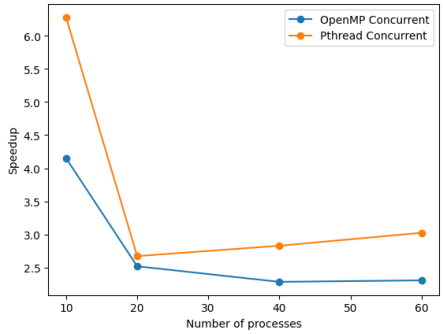

Рисунок 6 – Коэффициент ускорения вещания с линейной блокировкой MPI при синхронизации OpenMP и Pthread

Из рисунка 6 видно, что многопоточная передача данных значительно улучшает производительность по сравнению с однопоточной передачей.

MPI_Bcast, встроенная функция в MPI, выполняет передачу данных быстрее с увеличением числа процессов благодаря использованию более эффективных алгоритмов.

На рисунке 7 показано, что ускорение OpenMP и Pthreads наибольшее в начале, но Pthreads достигает минимума при 20 процессах, а затем ускорение OpenMP стабилизируется около значения 2. По закону Амдаля ускорение Pthreads должно приближаться к 2, так как мы выделили два ядра процессора корневому процессу.

В модели линейного распространения многопоточная неблокирующая передача и прием данных показывают лучшую производительность по сравнению с многопоточной блокирующей передачей и приемом на рисунке 7.

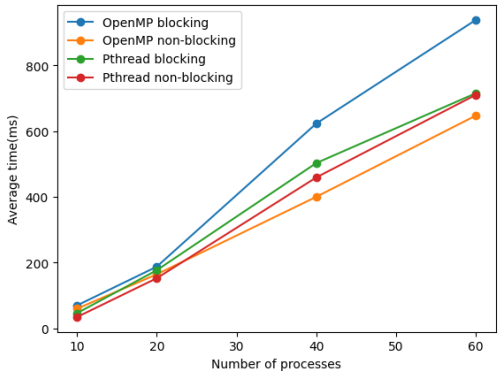

Рисунок 30 – Блокировка и неблокировка при многопоточном линейном вещании

5. Эффект от использования кластера в достижении целей работы:

Использование кластера NSU позволило повысить производительность и параллельную обработку задач. Кластер состоял из 48 серверов HP BL2x220c G7 с двухузловой архитектурой. В работе были исследованы различные методы параллельных вычислений, включая грубозернистую и мелкозернистую параллельность, а также режимы коммуникации и синхронизации. Были проведены оптимизации времени выполнения матричных операций с использованием комбинации MPI, OpenMP и Pthreads. Также были изучены и проанализированы различные режимы коммуникации, включая блокирующую и неблокирующую коммуникацию в MPI, а также линейное распространение данных. Общий эффект от использования кластера заключается в повышении производительности и ускорении вычислений путем распараллеливания и оптимизации коммуникации. Это позволило достичь поставленных целей исследования и получить более эффективные результаты.