Санкт-Петербургский государственный университет

Программная инженерия

Группа 24.М71-мм

Оптимизация производительности пакета BmnRoot

Ли Цзишэн

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры вычислительной физики, к. ф.-м. н., Немнюгин С. А.

Санкт-Петербург
2025

Оглавление

Введение

BM@N (Baryon Matter at Nuclotron) - первый эксперимент по ядерной физике с фиксированной мишенью в рамках программы исследований на ускорительном комплексе NICA, и его успешная реализация в значительной степени зависит от эффективной и надежной программной поддержки[1]. Пакет BmnRoot, являющийся основным инструментом обработки данных для эксперимента BM@N, основан на среде CERN ROOT, библиотеке моделирования Geant4 и объектно-ориентированной среде FairRoot. предоставляет мощные инструменты для исследования работы детектора, моделирования событий, реконструкции примеров и анализа физических данных, а также для выполнения других функций[2]. С помощью BmnRoot исследователи могут моделировать и калибровать подсистемы детектора BM@N, а также реконструировать и анализировать события столкновений, тем самым поддерживая задачи экспериментальной физики.

Однако текущее программное обеспечение BmnRoot имеет ряд проблем с точки зрения вычислительной производительности. Во-первых, существующий фреймворк обнаруживает узкие места в вычислительной производительности при больших наборах данных и высоких требованиях к пропускной способности. Когда Geant4, ROOT и FairRoot последовательно ввели поддержку многопоточности (например, Geant4MT), код BmnRoot пришлось модифицировать, чтобы адаптировать к требованиям многопоточной параллельной работы[2]. Это говорит о том, что отсутствие распараллеливания является одной из причин длительного времени вычислений. Во-вторых, есть недостатки и в управлении памятью. При длительном крупномасштабном моделировании объем памяти BmnRoot может постоянно увеличиваться, могут возникать проблемы с несвоевременным освобождением ресурсов или утечкой памяти. Подобные проблемы с управлением памятью могут привести к излишнему расходу ресурсов и снижению стабильности программы.

Поэтому проведение оптимизационных исследований узких мест

производительности и проблем управления памятью в BmnRoot имеет большое практическое значение и прикладную ценность.

1. Постановка задачи

Основная цель этой практики - сфокусироваться на программном фреймворке BmnRoot и использовать методы высокопроизводительных вычислений для настройки его производительности, что включает в себя:

1. Выявление «горячих точек» производительности: использование инструментов анализа для выявления «горячих точек» производительности в коде BmnRoot, определение основных узких мест в вычислительных функциях или модулях, а также определение направления оптимизации.

2. Оптимизация эффективности работы кода: для выявленных «горячих точек» используются соответствующие методы для оптимизации реализации кода, сокращения избыточных вычислений и ненужных накладных расходов, повышения эффективности работы программы.

3. Улучшение управления памятью: оптимизируйте структуру использования памяти BmnRoot, устраните потенциальные утечки памяти, избегайте ненужного выделения и копирования памяти, уменьшите занимаемую память и повысьте эффективность ее использования.

Ожидается, что достижение вышеуказанных целей позволит значительно повысить производительность BmnRoot при моделировании и обработке данных, обеспечив более эффективный инструмент для поддержки последующего анализа данных эксперимента BM@N.

2. Обзор

2.1. Высокопроизводительные вычисления в физике высоких энергий

Высокопроизводительные вычисления относятся к методам решения больших и сложных задач с помощью объединенных мощных вычислительных ресурсов. Их цель - ускорить выполнение вычислительных процедур за счет распараллеливания и интеграции ресурсов, тем самым сократив научный рабочий процесс. В области физики высоких энергий, где объем данных, получаемых в ходе экспериментов, чрезвычайно велик, а вычислительные задачи моделирования и обработки данных тяжелы, технология высокопроизводительных вычислений(HPC, High Performance Computing) стала незаменимым инструментом. Кластеры HPC обычно состоят из большого количества взаимосвязанных многоядерных узлов, которые используются для одновременной обработки огромных объемов данных или событий с помощью параллельных вычислений. В результате программное обеспечение для высокопроизводительных вычислений должно быть оптимизировано для параллельных архитектур, чтобы наилучшим образом использовать эти вычислительные ресурсы. В аналогичных областях оптимизация программного обеспечения для высокопроизводительных вычислений достигла значительных результатов, например, в литературе [3] было достигнуто 15,52-кратное ускорение на платформах CPU и 17,14-кратное ускорение на платформах GPU, используя 16-поточную версию OpenMP в качестве базовой.

2.2. Обзор методов оптимизации

Оптимизация производительности программного обеспечения для физики высоких энергий может быть реализована на нескольких уровнях, включая оптимизацией при компиляции, параллельные вычисления и методы гетерогенных вычислений. Во-первых, оптимизация компиляции позволяет использовать передовые методы оптими-

зации, предоставляемые компиляторами для повышения эффективности кода. Современные компиляторы C/C++ (например, GCC, LLVM) предоставляют множество уровней и опций оптимизации, например, использование уровня оптимизации -O3 в сочетании с директивой -march=native позволяет компилятору выполнять более агрессивную оптимизацию для целевой архитектуры, включая автоматическую SIMD-векторизацию[4]. С помощью этих опций компиляции к коду автоматически применяются такие оптимизации, как разворачивание циклов и инлайнинг функций, что помогает сократить количество инструкций и накладные расходы на доступ к памяти. Во-вторых, параллельные вычисления - это основной способ повышения производительности трудоемких вычислительных задач. Распространенные модели параллельных вычислений включают параллелизм с общей памятью и параллелизм с распределенной памятью. OpenMP - это стандартный интерфейс для многопоточного параллелизма с общей памятью, который позволяет программам выполнять многопоточные задачи одновременно на многоядерных процессорах. MPI (Message Passing Interface) используется в средах с распределенной памятью, где параллелизм достигается за счет передачи сообщений между различными процессами. На многоузловом кластере производительность MPI-программ почти линейно зависит от количества вычислительных узлов. На практике OpenMP и MPI часто комбинируют для построения гибридных параллельных моделей, чтобы использовать вычислительную мощь как одноузловых многоядерных, так и многоузловых кластеров для лучшей масштабируемости. Многие рабочие нагрузки в физике высоких энергий естественным образом подходят для параллельных вычислений, например, моделирование и реконструкция событий часто могут быть выполнены в параллелизме на уровне событий, поскольку различные события независимы друг от друга и могут обрабатываться параллельно отдельными потоками или процессами. Из литературы известно, что внедрение параллелизма на уровне событий в BmnRoot приводит к значительному увеличению коэффициента ускорения и масштабируемости моделирования по мере увеличения количества потоков и событий[2].

Во-вторых, гетерогенные вычисления на GPU также получили широкое внимание в физике высоких энергий в последние годы. Имея сотни ядер для потоковой обработки, GPU подходят для ускорения вычислительно-интенсивных задач с параллелизмом данных. Благодаря таким технологиям, как CUDA или OpenCL, такие вычисления, как моделирование слежения, обработка изображений или машинное обучение, могут быть переложены на GPU для выполнения, что значительно увеличивает пропускную способность.

Однако важно отметить, что реальный эффект от ускорения GPU зависит от характеристик алгоритма и объема данных. Если вычисления требуют частой передачи данных между CPU и GPU, накладные расходы на передачу данных могут нивелировать преимущества параллелизма GPU. Для небольших задач или задач с интенсивным доступом к памяти иногда более эффективным оказывается многопоточный параллелизм чистого CPU. Поэтому при внедрении вычислений на GPU необходимо взвесить детализацию задачи и характеристики пропускной способности данных, чтобы выбрать подходящую схему ускорения вычислений.

2.3. Опыт оптимизации с помощью ROOT и FairRoot

В области физики высоких энергий существует множество развитых программных фреймворков, накопивших богатый опыт оптимизации производительности, а фреймворк ROOT, разработанный в ЦЕРНе и являющийся де-факто стандартным инструментом для анализа данных, не только хорошо оптимизирован с точки зрения ввода-вывода и построения графиков, но и внедрил новые технологии, такие как многопоточный параллельный анализ (например, параллельное заполнение гистограмм с помощью Intel TBB), векторные вычисления (например, библиотека VECCore) и другие новые технологии, чтобы справиться с растущими требованиями к обработке данных на БАК. библиотека VECCore) и другие новые методы, позволяющие справиться с растущи-

ми требованиями БАК к обработке данных. Это показывает, что, улучшая алгоритмы и используя все преимущества аппаратных возможностей, традиционное программное обеспечение HEP также может получить значительное ускорение.FairRoot, как экспериментальный фреймворк на основе ROOT, был принят многими экспериментами, включая BM@N, с акцентом на модульность и параллелизм, FairRoot уже поддерживает многопоточные симуляции Geant4 и параллельные потоки обработки данных. Параллельные возможности FairRoot служат прямым ориентиром при разработке программного обеспечения BM@N: с появлением версии Geant4 MT и интерфейса Virtual Monte Carlo (VMC) фреймворк FairRoot был обновлен для поддержки многопоточности, что побудило команду разработчиков BM@N модифицировать код BmnRoot для совместимости и использования преимуществ этих параллельных возможностей[2]. В совокупности, использование опыта оптимизации этих программ (например, использование параллельных функций базовых библиотек, оптимизация управления памятью и алгоритмические улучшения) поможет направить практику оптимизации производительности BmnRoot.

3. Описание решения

3.1. Анализ программного кода BmnRoot

Сначала было проведено комплексное профилирование производительности существующего кода BmnRoot. Программное обеспечение запускалось и анализировалось с помощью инструментов анализа производительности, таких как GNU gprof, для выявления наиболее трудоемких участков выполнения программы, то есть «горячих точек» производительности[5]. Основные «узкие места» — функции и алгоритмические сегменты — выявляются путем анализа графиков вызовов и отчетов о временном потреблении функций, генерируемых gprof. В то же время инструменты анализа памяти, такие как Valgrind, используются для определения использования памяти программой. Инструмент Memcheck от Valgrind способен выявить такие ошибки, как некорректные чтения и записи в память, неосвобожденная память, а также сообщить о потенциальных утечках памяти[6]. С помощью этих инструментов анализа выясняются конкретные проблемы и узкие места в процессорных вычислениях и управлении памятью BmnRoot, что создает основу для последующей оптимизации.

3.2. Оптимизация компиляции

Следует использовать все преимущества опций оптимизации компилятора для повышения эффективности кода без изменения алгоритма. При перекомпиляции BmnRoot с помощью компиляторов GCC или LLVM включаются флаги расширенной оптимизации (например, -O3, -Ofast и т. д.), а также оптимизация набора инструкций для целевой платформы (например, -march=native). Эти опции позволяют компилятору автоматически применять более агрессивные стратегии оптимизации, такие как разворачивание циклов, инлайнинг функций и автоматическая векторизация[4].

3.3. Параллельная оптимизация

Для распараллеливаемых задач в BmnRoot применяются методы многопоточности и многопроцессорности для улучшения параллелизма. Во-первых, с точки зрения многопоточного параллелизма, OpenMP используется для распараллеливания горячих вычислительных ядер. Например, инструкции OpenMP добавляются в трудоемкие модули, такие как циклы для экземпляров или расчеты отслеживания траектории, чтобы добиться многопоточной одновременной обработки нескольких экземпляров или частиц. Поскольку события в физике высоких энергий обычно независимы друг от друга, использование параллелизма на уровне событий хорошо подходит: каждый поток обрабатывает отдельное событие и способен линейно увеличить пропускную способность симуляции или реконструкции. Было показано, что такая параллельность обеспечивает хорошую масштабируемость в симуляторах BM@N[2]. Во-вторых, с точки зрения многопроцессного параллелизма, MPI можно использовать для разделения задач, если вычисления необходимо масштабировать на несколько узлов. Например, большое количество событий назначается нескольким процессам для отдельной обработки, каждый из которых выполняется на отдельном узле кластера. MPI отвечает за передачу необходимых данных (например, объединение конечных результатов) между процессами. Объединение MPI и OpenMP в гибридную параллельную модель позволяет полностью использовать несколько ядер каждого узла кластера, преодолевая при этом ограничения памяти одной машины для достижения больших масштабов параллельных вычислений. После оптимизации параллельных вычислений эффект от распараллеливания оценивается путем тестирования коэффициента ускорения и эффективности программы при различном количестве потоков и процессов, а также настройки разделения нагрузки для достижения оптимальной параллельной эффективности.

3.4. Оптимизация ускорения GPU

Для ключевых модулей BmnRoot, требующих больших вычислительных затрат и обладающих параллелизмом данных, изучается возможность использования GPU для ускорения. Необходимо выбрать алгоритмы, подходящие для вычислений на GPU (например, матричные операции в анализе данных), затем надо реализовать соответствующие функции ядра для GPU, используя NVIDIA CUDA или OpenCL. Выгрузите эти вычислительные ядра на GPU для параллельного выполнения, чтобы воспользоваться массово параллельной вычислительной мощностью GPU. В процессе реализации необходимо обработать передачу данных между хостом (CPU) и устройством (GPU) и максимально использовать данные, находящиеся в памяти, чтобы уменьшить накладные расходы на передачу. Литература показывает, что параллелизм GPU может обеспечить значительное ускорение, если гранулярность задачи достаточно велика; однако для слишком мелких задач частое копирование данных может сделать выигрыш от ускорения GPU незначительным. Поэтому в данном исследовании мы оценим эффективность GPU-ускорения после тестирования, чтобы решить, в каких частях BmnRoot следует применить GPU-оптимизацию. Если GPU-оптимизация возможна, то влияние различных конфигураций потоков GPU и шаблонов доступа к памяти на производительность будет сравниваться для настройки реализации ядра CUDA. Фактические преимущества GPU-ускорения будут оценены в сравнительных экспериментах с многопоточной версией для CPU.

3.5. Оптимизация управления памятью

Наряду с оптимизацией производительности, этот проект будет направлен на улучшение эффективности использования памяти в BmnRoot. Во-первых, на основе результатов вышеупомянутых анализов будут найдены участки кода, где происходят утечки памяти или дублирование выделений. На этом практике планируется использовать Valgrind для анализа производительности программы. Это мощный ин-

струмент отладки памяти и анализа производительности, с помощью которого можно обнаружить утечки памяти, выход за границы массива, неинициализированные переменные и т. д. и создать наглядные отчеты. Используя отчеты Valgrind об осмотре памяти и результаты анализа, можно определить, какие объекты не освобождаются своевременно или какие структуры данных занимают слишком много памяти[6]. Впоследствии код должен быть рефакторингован и исправлен: каждое динамическое выделение памяти должно сопровождаться соответствующим освобождением, чтобы исключить рост объема памяти из-за накопления утечек памяти[4]. В то же время структуры данных оптимизируются для экономии памяти, например, избегая ненужных копий данных, повторно используя уже выделенные пулы памяти, уменьшая генерацию временных объектов и т. д. Если некоторые алгоритмы в настоящее время обменивают большой объем памяти на скорость вычислений (пространство на время), необходимо рассмотреть также возможность внедрения более эффективных стратегий управления памятью, чтобы уменьшить объем памяти при сохранении производительности. Для часто используемых данных в памяти следует обратить внимание на паттерны доступа к ним, чтобы улучшить показатели попадания в кэш, например, скорректировать выравнивание и расположение данных в памяти, а также использовать структурное заполнение памяти для уменьшения псевдообмена. При необходимости замените политику распределения по умолчанию на более эффективный контейнер или распределитель пула памяти. Помимо ручной оптимизации, в данном исследовании мы также оценим эффективность оптимизации памяти с помощью инструментов. Запустив такие инструменты, как Valgrind, и сравнив профили использования памяти и мусорного пространства до и после оптимизации, можно проверить, устранены ли утечки памяти, а также снижено ли пиковое использование памяти[7]. Ожидается, что с помощью ряда мер по оптимизации памяти можно значительно сократить ненужные затраты памяти BmnRoot и повысить стабильность программы при длительной работе.

Заключение

Ожидаемые результаты практики следующие:

1. Оптимизированная версия программного обеспечения BmnRoot: создается оптимизированная по производительности версия программного обеспечения BmnRoot. Показатели производительности до и после оптимизации будут сравниваться с помощью тщательного бенчмаркинга, при этом ожидается значительное улучшение скорости вычислений и использования ресурсов. Например, при одинаковом аппаратном окружении общее время обработки определенного количества событий оптимизированной версией значительно сократится, а при использовании многопоточного параллелизма можно получить почти линейное ускорение и хорошую масштабируемость при увеличении числа потоков[2]. В то же время оптимизация памяти должна привести к уменьшению пикового объема памяти, занимаемого программой, и длительные пакетные вычисления больше не будут страдать от постоянного роста памяти. Все эти улучшения будут проверены количественными экспериментальными данными, чтобы убедиться в реальном эффекте оптимизации.

2. Подробный отчет об оптимизации: будет написан полный технический отчет об оптимизационном решении BmnRoot. В отчете будет описан процесс анализа узких мест производительности, детали реализации различных мер по оптимизации, а также результаты сравнения производительности до и после оптимизации. Отчет будет включать ключевые изменения кода, возникшие проблемы и их решения. Графики и данные будут использоваться для того, чтобы показать, насколько оптимизация повысила производительность, и проанализировать, какие оптимизации внесли наиболее значительный вклад, а также возможности для дальнейшей оптимизации.

Список литературы

[1] Software Development for the BM@N Experiment at NICA / Konstantin Gertsenberger, Sergey Merts, Ilnur Gabdrakhmanov et al. // EPJ Web of Conferences. — Vol. 226. — Dubna, Russia : EDP Sciences, 2020. — P. 03008. — Mathematical Modeling and Computational Physics 2019. URL: https://doi.org/10.1051/epjconf/202022603008.

[2] Multithreaded Event Simulation in the BmnRoot Package / M. M. Stepanova, A. V. Driuk, S. P. Mertz et al. // Proceedings of the 9th International Conference "Distributed Computing and Grid Technologies in Science and Education" (GRID'2021). — Dubna, Russia : Joint Institute for Nuclear Research, 2021. — July 5-9. — P. 58–63. — URL: http://nica.jinr.ru.

[3] Wei Jianwen, Xu Zhigeng et al. Optimization of Metagenomic Gene Clustering on Heterogeneous Clusters // High Performance Computing Development and Application. — 2015. — Vol. 15, no. 4. — P. 54–59. — URL: https://www.ssc.net.cn/file/computing/2015/15_4.pdf.

[4] Norvig Paul. Optimizing C Code for HPC. — URL: https://www.paulnorvig.com/guides/optimizing-c-code-for-hpc.html (дата обращения: 17 февраля 2025 г.).

[5] URL: https://hpc.llnl.gov/ (дата обращения: 17 февраля 2025 г.).

[6] URL: https://51degrees.com/ (дата обращения: 17 февраля 2025 г.).

[7] Optimization of Software on High Performance Computing Platforms for the LUX-ZEPLIN Dark Matter Experiment / Venkitesh Ayyar, Wahid Bhimji, Maria Elena Monzani et al. // EPJ Web of Conferences. — Vol. 245. — Berkeley, California, USA : EDP Sciences, 2020. — P. 05012. — CHEP 2019. URL: https://doi.org/10.1051/epjconf/202024505012.