Отчет по оптимизации проекта для биржевой торговли

Введение

В рамках образовательного проекта по высокочастотной торговле (HFT) была поставлена задача оптимизировать обработку рыночных данных с целью минимизации задержек и повышения производительности системы. Основное внимание уделялось эффективной обработке обновлений ордербука, вычислению VWAP и сериализации данных.

Ограничения и условия

- Ограничения:
  - Нельзя было изменять runner.
  - Работа велась только с публичным фрагментом pcap-записи.
  - Временных и ресурсных ограничений не было.
- Метрики производительности:
  - Использовался perf_event_open для измерения следующих показателей:
    - cycles
    - cache-misses
    - branch-misses
  - CI-пайплайн от организаторов предоставлял итоговые замеры задержек (latency).

Этапы оптимизации

1. Базовая оптимизация

- Инициализация данных:
  - Исключение ненужных инициализаций и копирований данных.
- Улучшение структуры кода:
  - Минимизация количества временных объектов.
  - Использование std::move вместо итеративного копирования

2. Протокол специфичные оптимизации.

- Исключение ненужных в рамках задания операций, вроде валидации чек сумм, фильтрация пакетов по ip, проверки апдейтов на валидность, проверки vint данных на валидность, и т.д.
- Кеширование vint кодов instrument_id на старте программы, и дальнейшее побайтовое сравнение instrument_id вместо постоянного декодирования.
- Использования более оптимальных методов для декодирования vint, с учетом того что размер данных в байтах по сути известен в компайл тайм.
- Проверка на summary-поля только в определенном месте, в конце группы, таким образом экономия бранчей.
- Использования структуры данных фиксированной длины для парсинга апдейтов, определив максимальный размер для числа групп, инкрементов и т.д. на тестовых данных.

3. Алгоритмические оптимизации

- Поиск ордербука по instrument_id:
    вначале был использован unordered_map, позже он был заменеу на статический массив и бинарный поиск по нему, позже бинарный поиск был заменен на линейный, снизив количество бранч миссов. Далее random access подход был заменен на статический массив длины 2^16, где instrument_id - индекс в массиве. Далее, в угоду лучшей кеш локальности, был выбран комбинированный подход, где с помощью std::bitset с instrument_id в качестве индекса проверяется нужный ли это инструмент, и далее с помощью векторизованного поиска в массиве нужных иструментов, в одну инструкцию, находится ордербук.
- Хранение оффсетов в ордербуке, вместо постоянного подсчета актуальных цен.

4. Оптимизации с использованием SIMD.

- Значительный эффект дало использование векторных инструкций для поиска нужный группы в инкрементальном апдейте, вместо парсинга с помощью прыжков в цикле на длину поля. Уменьшение score ~30k.
- Были также проведены эксперименты с SIMD для обновления ордербука, для подсчета WVAP, а также для декодирования vint, однако это не дало улучшения в измеряемых метриках, поэтому они были заменены на более читабильные конструкции.
- Кроме этого, были проделаны некоторые теоретические исследования, в ходе которых обнаружилось, что на более свежих версиях sapphire rapids доступна AVX-512 инструкция intersect, с помощью которой можно было бы совершить поиск не по header_id, а сразу по instrument id за один такт на батче данных размером 64 байт, что снизило бы количество if statement на один апдейт, по грубым подсчетом снижая итоговый score еще на 20-30к.

5. Оптимизация memory layout.

- Выравнивания массивов и буферов по размеру кэш-линии, над которыми потенциально применяются векторные инструкции, и частое чтение.
- С помощью экспериментов на тестовых данных были выявлены верхние границы, для числа инструментов, максимального кол-ва уровней в ордербуке в процессе апдейта, максимального количества кэшированных апдейтов, в процессе рекавери. Благодаря этому все аллокации производятся на старте программы, оптимизируя тем самым hoth-path, а также размер структур данных сводится к минимально возможному, потенциально улучшая cache hit/miss ratio .
- Активное использования префетчинга при чтении апдейта.

6. Оптимизации на этапе компиляции.

- Для Release сборки использовались опции `-march=native -mtune=native -ffast-math -flto -fno-exceptions -fno-rtti -fomit-frame-pointer -DNDEBUG`
- Были проведены эксперименты с разными `march, однако более оптимального варианта найдено не было.`
- Попытка использования profile-guided optimization (PGO).
- Отказ от PGO из-за увеличения количества ошибок предсказания ветвлений и ухудшения производительности.

7. Оптимизация работы с SPSC queue.

- Zero-copy парсинг пакетов, напрямую из shared memory. Для снижения cache misses, бы использован префетчинг, что дало заметный результат, ~15к снижение score.
- Использование отложенного store для consumer offset. Поскольку раннеру не обязательно знать что мы прочитали пакет до отправки ему ответу, можно делать запись в атомарную переменную уже после отправки ответа.

8. Не упомянутые оптимизации.
- Использования loop unrolling там где возможно.
- Инлайнинг небольших функций, где нет heavy бранчинга.
- Активное использование attributes для подсказок компилятору, какой бранч лежит в hoth path.
- Использование compile time evaluation (constepr and consteval).
- Активное использование short-circuit optimization.
- Предпочтение value семантики reference семантике, если размер типа данных аргумента меньше 64 бит.
- Активное использование ключевого слова const для неизменяемых данных, благодаря чему компилятор и cpu пайплайн могут производить более агрессивные оптимизации исполнения.

Заключение

В ходе работы были применены различные техники оптимизации, начиная от базовых улучшений структуры кода и заканчивая использованием расширенных инструкций AVX-512 для векторизации вычислений. Профилирование с использованием perf_event_open позволило выявить узкие места и эффективно их устранить. Несмотря на попытку использования PGO, было принято решение отказаться от него из-за негативного влияния на производительность.

Данные оптимизации привели к значительному улучшению производительности системы обработки рыночных данных, что подтверждается снижением задержек и улучшением метрик, собранных с помощью инструментов профилирования.