1 слайд

Здравствуйте, уважаемая комиссия. Сегодня, я, хочу представить вам свою выпускную работу. Тема моей работы: Улучшение производительности программного кода на C++ заменой контейнеров std::list на контейнеры std::vector.

2 слайд

Цель работы: заключается в поиске оптимальной стратегии замен контейнера «list» контейнером «vector» на участках последовательного обращения к элементам массива при условии, что требуемые для её реализации дополнительные ресурсы ОП не превысят верхней границы.

Задача работы: создание приложения, программно реализующего алгоритм оптимизации. Экспериментальная проверка эффективности.

Данный вид оптимизации даёт существенный прирост производительности на машинах с большим объёмом ОП.

В принципе, что такое оптимизация программы. Под оптимизацией программы подразумевают такие преобразования, в результате которых она становится более эффективной, т.е. становится более экономной по памяти или/и более быстрой по выполнению тех же функций, что и до оптимизации.

Оптимизация проводится по двум частным критериям: время выполнения программы и объём памяти, которую она использует. Но эти два критерия противоречат друг другу, т.к. чтобы уменьшить время работы, необходимо увеличить размер потребляемой памяти и наоборот.

3 слайд

Моя оптимизация основана на особенностях работы контейнеров vector и list, при добавлении элементов и при последовательном обращении к элементам.

В частности, при добавлении данных в контейнер «list» выделяются данные для хранения одного элемента, а в контейнере «vector» выделение памяти происходит порционально: если текущий блок памяти, предназначенный контейнеру «vector» исчерпан, то при добавлении нового элемента будет зарезервирован новый блок памяти, размер которого на 50% больше прежнего. Таким образом, при одном и том же количестве элементов фактически размер занимаемой памяти между контейнерами «list» и «vector» может различаться до 1.5 раз (КОД).

Что касается последовательного доступа к элементам, переход к элементу в vector происходит в течение одной операции перемещения по индексу, а аналогичное действие в контейнере «list» выполняется гораздо медленнее из-за того, что данные в списке расположены не последовательно, а в произвольных местах динамической памяти и связаны между собой с помощью специальных указателей.

В связи с вышесказанным актуальной является задача замены на участках последовательного поиска контейнеров list на контейнеры vector с учётом ограничения на объём ОП необходимой для такой оптимизации. Как было сказано выше, контейнеру vector может потребоваться до 1,5 раз больше ОП, чем list, из-за порционального выделения новых блоков.

4 слайд

В данной работе формулируется оптимизационная задача максимизирующая суммарный выигрыш во времени последовательного доступа к элементам данных, путём замены контейнера «list» на контейнер «vector».

–суммарный выигрыш, получаемый от замен на участках доступа контейнеров list на контейнер vector;

–количество рассматриваемых контейнеров;

– булева переменная, которая определяет, будет ли использоваться i – ый контейнеры для оптимизации;

– количество элементов i – го контейнера;

- среднее время доступа одного элемента в i–ый контейнер типа list, измеряется в секундах;

5 слайд

- среднее время доступа одного элемента в i–ый контейнер типа vector, измеряется в секундах;

– размер одного элемента i – го контейнера, измеряется в байтах;

V– Верхняя граница дополнительного объёма оперативной памяти ЭВМ, предназначенной для размещения данных, измеряется в байтах.

В свою очередь:

, где – средняя скорость доступа данных в контейнер типа list. Измеряется в байт/секунды.

, где – средняя скорость доступа данных в контейнер типа vector. Измеряется в байт/секунды.

Задача заключается в том, чтобы оптимизировать неэффективные участки кода программы, с учётом введённого ограничения на память. Для этого производился анализ кода, замена и проверка.

6 слайд

С доски алгоритм поиска решения

1. Получаем обработанный текст;
2. Начинаем рассматривать полученные данные;
3. Генерируем полный перебор;
4. Рассматриваем каждую комбинацию, которая состоит из 0 и 1, где 0 – замена не производится, 1 – замена производится, и длина каждой последовательности равна количеству не оптимально выбранным контейнерам;
5. Из множества решений выбираем то, которое максимизирует целевую функцию и удовлетворяет ограничениям.

7 – 8 слайды

Пример решения задачи вручную

* 1. Анализируем текст программы и находим, возможно, неоптимальный код. В нашем случае мы считаем, что it, it3 и it4 не оптимальны. То есть контейнеры, которые используют эти итераторы, могут быть подобраны программистом неправильно;
  2. Определяем тип этих контейнеров, для того, чтобы корректно высчитать целевую функцию;
  3. Размерность каждого контейнера: 5 , 3 и 10 элементов соответственно;
  4. Полным перебором подбираем оптимальную комбинацию из наших контейнеров:
* 0, 0, 0 – целевая функция равна 0;
* 0, 0, 1 – целевая функция равна 40, ресурсов затрачено: 120 байт;
* 0, 1, 0 – целевая функция равна 6, ресурсов затрачено: 18 байт;
* 0, 1, 1 – целевая функция равна 46, ресурсов затрачено: 138 байт;
* 1, 0, 0 – целевая функция равна 10, ресурсов затрачено: 30 байт;
* 1, 0, 1 – целевая функция равна 50, ресурсов затрачено: 150 байт;
* 1, 1, 0 – целевая функция равна 16, ресурсов затрачено: 48 байт;
* 1, 1, 1 – целевая функция равна 56, ресурсов затрачено: 168 байт;

Таким образом, наилучшей комбинацией считается 1, 0, 1. Это означает, что если мы изменим тип у контейнеров numbers1 и numbers4 с list на vector, то увеличим скорость и не выйдем за ограничение.

9 слайд

1. Анализируем текст программы, выбранный пользователем;
2. Разбиваем по шаблону весь текст и находим необходимые блоки кода;
3. Определяем, возможно, неоптимальные участки кода, путём проверки: если в определённом блоке имеется прямое обращение к элементам, то, считаем, что это неоптимальный код, и мы будем его рассматривать.
4. Определяем тип рассматриваемых контейнеров;
5. Полным перебором определяем оптимальную комбинацию, т.е. комбинацию с максимальным выигрышем и удовлетворяющую введённому ограничению;

10 слайд

Для реализации алгоритма, описанного ранее, было разработано приложение на языке C#, платформы .NET 3.5. Использовалась технология Windows Forms. Интерфейс программы представляет собой окно, содержащее элементы управления.

Для загрузки файла с исходным кодом программы, написанной на C++, нажимаем на кнопку «Выберите файл». Откроется диалоговое окно, в котором нужно будет выбрать файл

11 слайд

Когда пользователь выберет файл, то его содержимое, будет выведено на экран

После выбора файла, пользователь должен нажать на кнопку «Начать анализ участка кода», т.е. программа должна проанализировать код и вывести соответствующие результаты на экран

12 слайд

После анализа кода, пользователь должен ввести некоторые данные в специальном окне «Данные для подсчёта выигрыша», такие, как: (по умолчанию 1), (по умолчанию 2), V, а так же в таблице необходимо заполнить столбец «Количество элементов»

Когда все данные введены, можно нажимать на кнопку «Автоматический расчёт». В результате полным перебором будет подобрана оптимальная стратегия с учётом целевой функции и ограничения. Решение будет отображено в окне «Данные для подсчёта выигрыша». Будет выведен суммарный выигрыш, а так же количество памяти, предназначенное для этой оптимизации.

В таблице, в первом столбце, будут отмечены те checkbox – ы, соответствующие контейнеры которых должны быть заменены. V и рекомендации.

13 слайд

Так же пользователь сам может выбирать элементы, вопреки тому, что вывела программа, но колонка «Рекомендации» остаётся неизменной, в случае если пользователь захочет использовать стратегию, предложенную программой.

14 слайд

Эксперимент заключался в оценке степени прироста производительности, получаемом в результате замены контейнера «list»контейнером «vector», на участках последовательного обращения ко всем элементам массива данных

Описание: в ходе эксперимента, размер контейнеров «list»и «vector»синхронно менялся от 1 000 000 элементов до 10 000 000 с шагом 1 000 000. Исходный кодтестового примера:

КОД СТОПЭ

15 слайд

Эксперимент показал, что замена контейнера «list»контейнером «vector», в местах последовательного обращения ко всем элементам массива даёт существенный прирост в производительности.

16 слайд

Эксперимент заключался в увеличение объёма текста программы и оценке производительности анализа кода.

Описание: В ходе эксперимента я увеличивал количество блоков, которые подлежали рассмотрению и засекал время, за которое программа проанализирует текст программы, написанной на C++.

График

Из графика видно, что при увеличении числа блоков неоптимизированного кода, увеличивается и время анализа программы.

17 слайд

Эксперимент заключался в оценке выбранного мной метода – полного перебора, т.е. увеличивалось количество контейнеров, которые необходимо было рассматривать и засекалось время нахождения решения.

Описание: При проведении эксперимента, я брал программу, написанную на C++, где был обнаружен только один блок для рассмотрения и в дальнейшем увеличивал их количество от 1 до 12. Во всех программах верхняя верхняя граница используемой для оптимизации ОП V = 500

График

Из графика видно, что, чем контейнеров, которые необходимо рассмотреть, в программе, написанной на C++, тем больше времени требуется для нахождения решения.

18 слайд

Для решения данной задачи, был разработан алгоритм замены и написана программа. А также проведены эксперименты и построены графики, обосновывающие цель данной работы и данной оптимизации в целом.

При использовании контейнеров C++ нужно иметь ввиду, как сильные стороны выбранного типа контейнера, так и его слабые стороны, и в зависимости от поставленных задач использовать каждый контейнер по назначению.

*Доверительный интервал* для среднего представляет интервал значений вокруг оценки, где с данным уровнем доверия (см. [*Элементарные понятия статистики*](http://statsoft.ru/home/textbook/esc.html)), находится "истинное" (неизвестное) среднее популяции. Например, если среднее выборки равно 23, а нижняя и верхняя границы доверительного интервала с уровнем *p*=.95 равны 19 и 27 соответственно, то можно заключить, что с вероятностью 95% интервал с границами 19 и 27 накрывает среднее популяции. Если вы установите больший уровень доверия, то интервал станет шире, поэтому возрастает вероятность, с которой он "накрывает" неизвестное среднее популяции, и наоборот. Хорошо известно, например, что чем "неопределенней" прогноз погоды (т.е. шире доверительный интервал), тем вероятнее он будет верным. Заметим, что ширина доверительного интервала зависит от объема или размера выборки, а также от разброса (изменчивости) данных. Увеличение размера выборки делает оценку среднего более надежной. Увеличение разброса наблюдаемых значений уменьшает надежность оценки (см. также [*Элементарные понятия статистики*](http://statsoft.ru/home/textbook/esc.html)). Вычисление доверительных интервалов основывается на предположении нормальности наблюдаемых величин. Если это предположение не выполнено, то оценка может оказаться плохой, особенно для малых выборок. При увеличении объема выборки, скажем, до 100 или более, качество оценки улучшается и без предположения нормальности выборки.